1993-03

Efficient scheduling of real-time compute-intensive periodic graphs on a large grain data flow multiprocessor

Akin, Cem

Monterey, California: Naval Postgraduate School

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION: UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT: Approved for public release; distribution is unlimited
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION: Computer Science Dept. Naval Postgraduate School; 6b. OFFICE SYMBOL (if applicable): CS	7a. NAME OF MONITORING ORGANIZATION: Naval Postgraduate School
6c. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000	7b. ADDRESS (City, State, and ZIP Code): Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION; 8b. OFFICE SYMBOL (if applicable)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State, and ZIP Code)	10. SOURCE OF FUNDING NUMBERS: PROGRAM ELEMENT NO. / PROJECT NO. / TASK NO. / WORK UNIT ACCESSION NO.

11. TITLE (Include Security Classification)
(U)Efficient Scheduling of Real-time Compute-intensive Periodic Graphs on a Large Grain Data Flow Multiprocessor

12. PERSONAL AUTHOR(S)
Akin, Cem

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Master's Thesis	FROM 05/92 TO 03/93	March 1993	171

16. SUPPLEMENTARY NOTATION: The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the United States Government.

17. COSATI CODES		
FIELD	GROUP	SUB-GROUP

18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
Large Grain Data Flow, Data Flow, Scheduling, Graph Restructuring, Run-time, Real-time, Compile-time, Resource Usage, Throughput

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

Architectures of computer systems based on Data Flow (DF) concepts attracted great attention as an alternative to conventional sequential architectures (Von Neumann). DF architectures are capable of efficiently exploiting a massive amount of parallelism inherent in many types of computation. They are programmed using directed graphs whose vertices are function modules and whose edges denote data dependencies between function modules. An important subclass of DF is Large Grain Data Flow (LGDF) which is efficiently used in computation intensive applications, such as signal processing. Presently, most leadoffs incorporate nondeterministic run-time technique to allocate system resources to support the execution (One such technique could be First Come First Served). Despite of the usual simplistic nature of scheduling techniques which, results in a low run-time overhead, the system throughput and predictability could rapidly degrade under high system load. To provide uniform output and improve the resource usage even under a high load, a compile-time technique called Revolving Cylinder (RC) was introduced. In this thesis, we present a LGDF simulator and a Graph restructurer that restructures the given graph according to the RC technique. We then perform a comparative experimental study of the different implemen-

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS	UNCLASSIFIED	
22a. NAME OF RESPONSIBLE INDIVIDUAL: Amr Zaky	22b. TELEPHONE (Include Area Code): (408) 656-2693	22c. OFFICE SYMBOL: CS/Az

tation of RC and the FCFS scheduling techniques. Our results demonstrate that there is a high potential for the RC technique, if a satisfactory node mapping technique is developed.

Approved for public release; distribution is unlimited

Efficient Scheduling of Real-time Compute-intensive Periodic Graphs on a Large Grain Data Flow Multiprocessor

by
Cem AKIN
LTJG, Turkish Navy
B.S., Turkish Naval Academy, 1987

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
March 1993

ABSTRACT

Architectures of computer systems based on Data Flow (DF) concepts attracted great attention as an alternative to conventional sequential architectures (Von Neumann). DF architectures are capable of efficiently exploiting a massive amount of parallelism inherent in many types of computation. They are programmed using directed graphs whose vertices are function modules and whose edges denote data dependencies between function modules. An important subclass of DF is Large Grain Data Flow (LGDF) which is efficiently used in computation intensive applications, such as signal processing. Presently, most leadoffs incorporate nondeterministic run-time technique to allocate system resources to support the execution (One such technique could be First Come First Served). Despite of the usual simplistic nature of scheduling techniques which, results in a low run-time overhead, the system throughput and predictability could rapidly degrade under high system load. To provide uniform output and improve the resource usage even under a high load, a compile-time technique called Revolving Cylinder (RC) was introduced. In this thesis, we present a LGDF simulator and a Graph restructurer that restructures the given graph according to the RC technique. We then perform a comparative experimental study of the different implementation of RC and the FCFS scheduling techniques. Our results demonstrate that there is a high potential for the RC technique, if a satisfactory node mapping technique is developed.

TABLE OF CONTENTS

I. INTRODUCTION

A. DATA FLOW ARCHITECTURES

The Data Flow (DF) architecture is an alternative to Von Neumann architecture and is capable of exploiting a massive amount of parallelism inherent in many types of computation. In the DF model of computation, a program is represented by a directed graph in which the nodes denote operations and the arcs connecting them represent data dependencies. Nodes become ready when their operands arrive, thus computations are data-driven. The DF model supports concurrent execution of ready nodes. In this model, execution of nodes is asynchronous because of its data-driven nature. Also, computations are free from side effects. There is no notion of shared data storage and results are conveyed directly by means of data arcs, [KARP 66]. These properties imply that multiprocessor architectures based on DF model need not suffer from the synchronization and coordination overheads incurred in Von Neumann architectures.

Depending upon the chosen granularity, DF architectures can be categorized in two groups: Large Grain Data Flow (LGDF) and Fine Grained Data Flow (FGDF) architectures. The granularity of nodes is crucial to the effectiveness of multiprocessing. For a given application, the larger the node grain size, the smaller the degree of parallelism that can be exploited. However, the larger the grain size, the smaller the amount of communication overhead that is incurred. Thus, fine granularity does not necessarily imply better performance, because it also increases communication overhead, which affects the parallelism exploited. The choice of granularity can be influenced by software engineering considerations as well as by parallel processing considerations, [LEE 89].

Real-time compute-intensive applications require predictable response time and high performance as measured by throughput. Satisfying response time and throughput requirements are critical for the correct functioning real-time applications. The predictability of both response time and throughput can be influenced by resource

allocation and communication overhead. Resource allocation and communication overhead can be controlled at compile-time and at run-time to lead to high throughput and deterministic response time.

Based on how a graph node and arc attributes are used at compile-time and how much control information is generated to aid the run-time mechanism, DF scheduling implementations can be classified as ***fully dynamic, self timed, static, fully static. Fully dynamic*** allocation performs all scheduling of nodes at run-time based on the readiness of inputs and resource availability. In a ***self timed*** allocation system, compiler determines the order of the node execution and allocates resources, but execution of the nodes is determined at run-time by data arrival. ***Static*** node allocation involves the assignment of a node to a processor, but the order of execution is left up to run-time scheduler based on the node's input data. In ***fully static*** allocation, the compiler determines the exact execution time assignment, and ordering of nodes based on that node's predicted behaviour, [LEE 90].

B. OBJECTIVES

First-Come-First-Served (FCFS) is a scheduling strategy where the ready node primitives are ordered by the time they become ready. High system loads may cause memory contention and imperfect computation/communication overlap resulting in a degradation of throughput and unpredictable response time. The FCFS strategy does not exert any run-time control to solve the above mentioned problem.

In this thesis, compile-time analysis of LGDF graphs is carried out using the Revolving Cylinder (RC) technique. RC restructures the original graph to obtain high throughput and predictable response time, [SLZ 92]. Performance analysis of results obtained by simulation is carried out to compare merits of different approaches.

1. Scope of the Thesis

In [LIT 91] and [SLZ 92], the RC approach to determine the node execution sequence was first suggested to enhance the system throughput.

This thesis refines the RC approach and its scope is analysis of previous work [LIT91] [BELL 92],development of an event driven simulator (PIPDAFS) for LGDF machines, and finally comparison of techniques for graph restructuring using the RC approach.

C. THESIS ORGANIZATION

Chapter II reviews the RC approach and the previous work accomplished in the area. In Chapter III, we present PIPDAFS (Periodic Inputs DAta Flow Simulator) and sample runs.In Chapter IV, we discuss and compare the graph restructuring techniques and analysis of the experimented results. In Chapter V, we conclude from the results along with suggestion for future work to be done.

II. BACKGROUND

A. REVOLVING CYLINDER ANALYSIS

The Revolving Cylinder (RC) technique for determining the node execution sequence performs compile-time analysis and results in the restructuring of the graph, [LIT 91], [SLZ 92]. RC technique restructures the application, described by a LGDF graph by using the machine configuration and the application graph as input.

The first step towards restructuring is to layout the graph nodes in a schedule that achieves some purpose (e.g. no nodes reading or writing from the same memory modules at the same time). The graphs that we deal with are Directed Acyclic Graphs (DAG). We benefit from research done on fine grained scheduling of parallel loops.

Loop scheduling literature [RAU 81] [HSU 86], has dealt with the problem of scheduling multiple iterations of the same parallel loop (represented by a DAG) to saturate the available resources. The resultant schedules possess a minimal initial cost for iterations, while possibly extending the completion time of each iteration.

Our scheduling technique can benefit from these results by noting that the real-time applications that we deal with require multiple instantiations of a DAG each of whose nodes is a large-grained primitive. These multiple instantiations have similarities with the execution of parallel iterations of a loop. Based on that, we drive the following scheduling principle[1]:

"The schedule(s) providing the maximal throughput with minimal initiation interval for a DAG depend only on the resource requirements of the DAG nodes and not on the topology of the DAG."

---

1. This principle can be obviously deduced from the work of [RAU 81] and [HSU 86] although it was not explicitly mentioned there.

This principle simply states that schedules with the same throughput can be found for different DAGs as long as they have similar sets of nodes with respect to the resource requirements. As a result of this principle, it might be worthwhile to reduce the DAG to be scheduled to an equivalent DAG with the same set of nodes V and an empty set of edges E before scheduling it as will be shown below.

The following examples should clarify the principle:

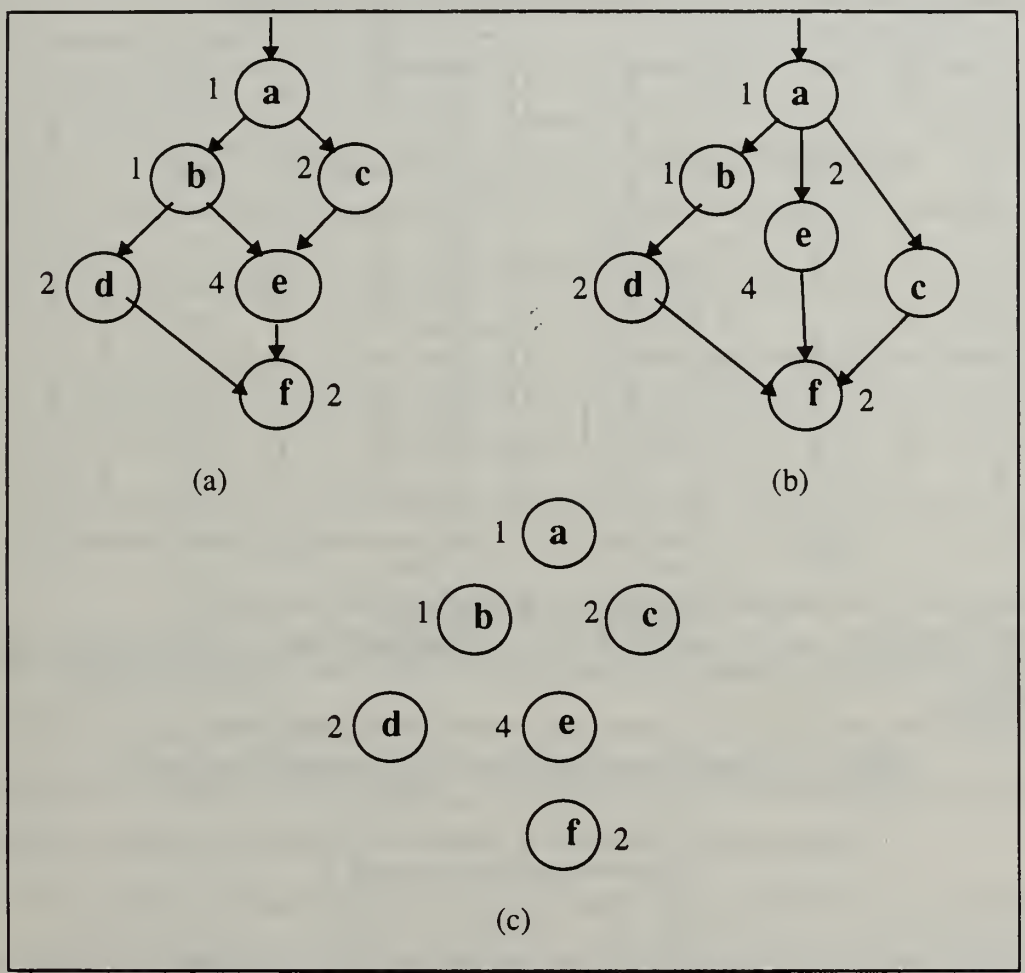


Figure 2.1. Sample Directed Acylic Graphs (DAG) (a)DAG I (b)DAG II (c)DAG III

The DAG I, DAG II and DAG III are three graphs with same node requirements and different edges. Then according to the above principle, they can have the same schedule which produces the same throughput. Two example of the many possible schedules for these graphs are shown in Figure 2.2a and Figure 2.2b, (The schedules assume two

processors).These examples depict the node execution pattern that is induced by a schedule. The nodes might belong to different instances. The exact instance to which a node belongs depends on the set of edges (that has been ignored so far) are in tables 1,2,3. We say nodes in the schedules with the indices of the instances they belong to. The above principle reduces optimal throughput of a DAG scheduling to a bin-packing problem. While the set of edges in a DAG has no effect on the potential throughput of the DAG, it will affect the instance response time associated with any schedule.

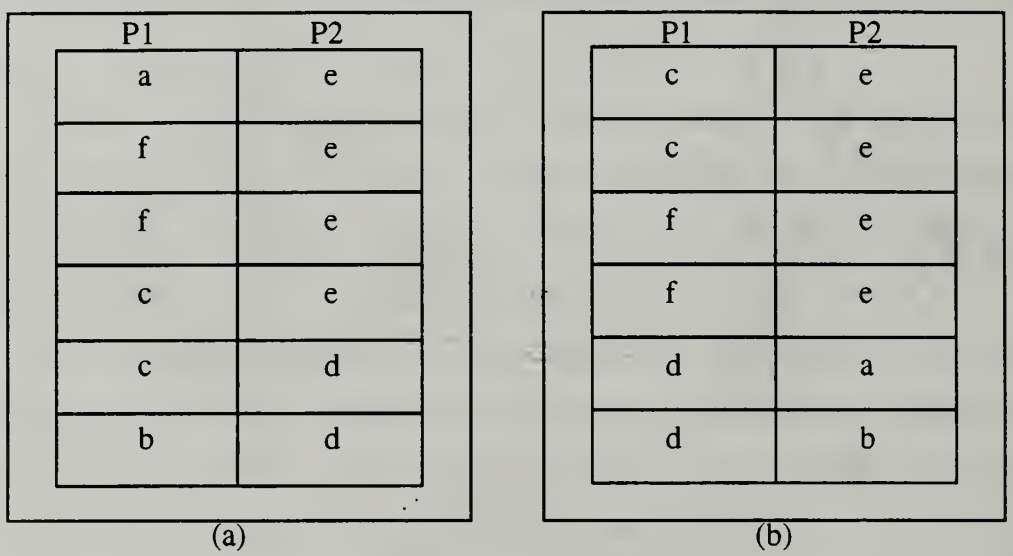


Figure 2.2. Two possible schedules for graphs with two processor

The schedules can be enforced as it is depicted Table 1, Table2 and Table 3 respectively.

Table 1: Compact Representation of the schedule I on DAG I

P1	P2
a_i	e_{i-1}
f_{i-2}	e_{i-1}
f_{i-2}	e_{i-1}
c_i	e_{i-1}
c_i	d_{i-1}
b_i	d_{i-1}

Table 2: Compact Representation of schedule II on DAG I

P1	P2
c_{i-1}	e_{i-2}
c_{i-1}	e_{i-2}
f_{i-3}	e_{i-2}
f_{i-3}	e_{i-2}
d_{i-1}	a_i
d_{i-1}	b_i

Table 3: Compact Representation of schedule I on DAG III

P1	P2
a_i	e_i
f_i	e_i
f_i	e_i
c_i	e_i
c_i	d_i
b_i	d_i

Consider the Tables 1 and 2 both of which show an indexed version of schedule I and II on DAG I. The corresponding schedule of one instance of DAG I using the Tables 1 and 2 are shown in Figure 2.3a and Figure 2.3b respectively. It is clear that these are different schedules possessing the property that they yield the same throughput.

The algorithms for assigning indices to the nodes in a schedule and to synchronize the nodes so as to be faithful to the schedule are the main purpose of "RC Scheduling". These algorithms can be found in, [SLZ 92].

The name "Revolving Cylinder" reflects a way looking at the schedule by wrapping the nodes around a cylinder, thereby causing its end to meet its beginning. For each node in the original graph, with the top and working toward the bottom, attempt to schedule the node at its earliest start time If it can not be inserted at that time, delay the start time by the

width of a slot and repeat until it can be inserted. The earliest start time of all descendants of that node and repeat the above sequence with the next node as the top node in the graph, [LIT 91] and [SLZ 92].

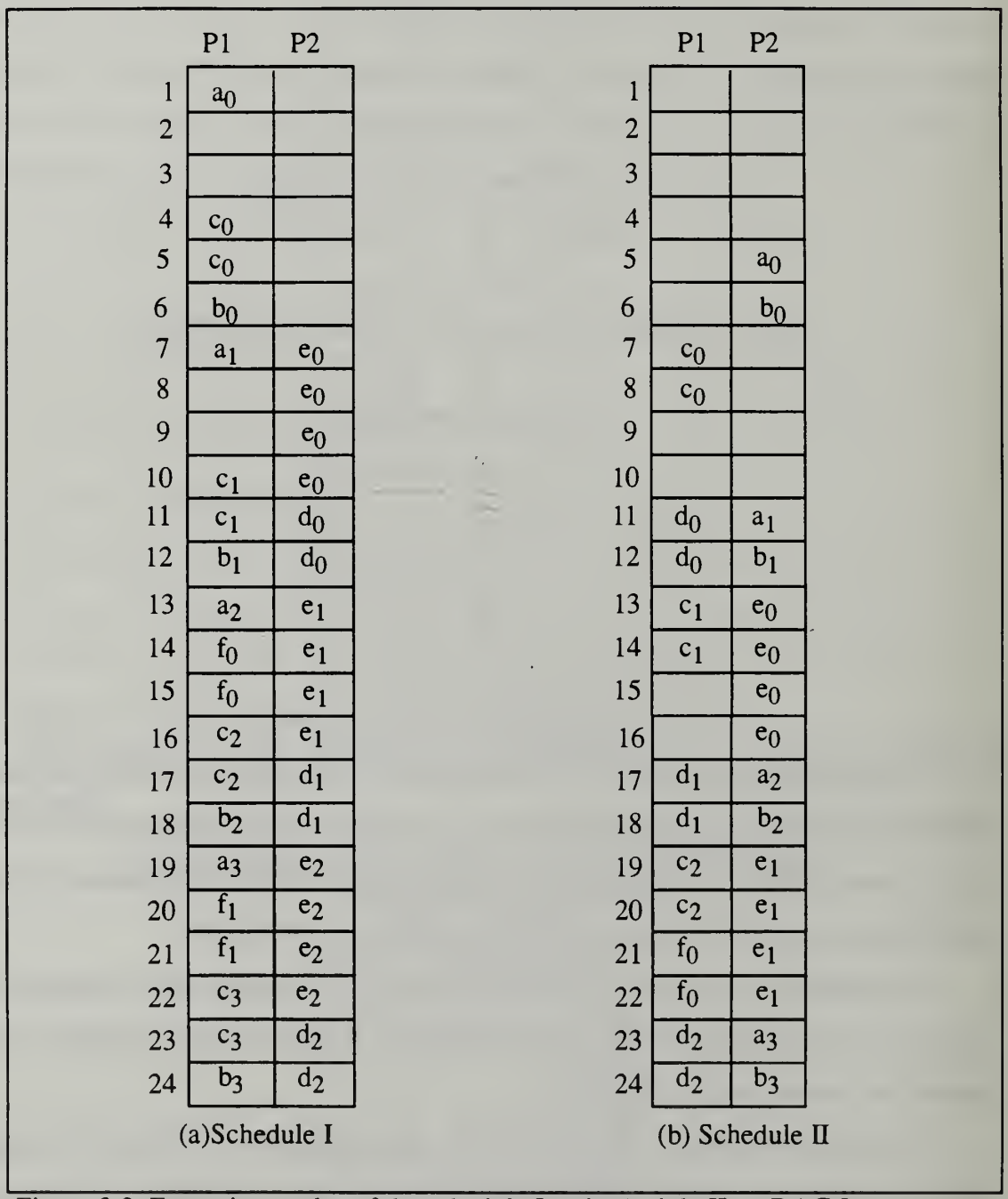


Figure 2.3. Execution cycles of the schedule I and schedule II on DAG I

In Figure 2.4, the execution of RC belonging to the DAG I is shown. Each node of the graph occupies a portion of the cylinder equal to its execution time. And a new instantiation could be started every six cycles, when two processor are used. Thus another instance of the graph can be overlapped with the first instance after six cycles. To prevent any conflict on the graph execution when instances are overlapped, nodes are assigned indices. For the example presented in Table 1, e_1 can not be executed at the same time as a_1 however, e_0 can, [SLZ 92].

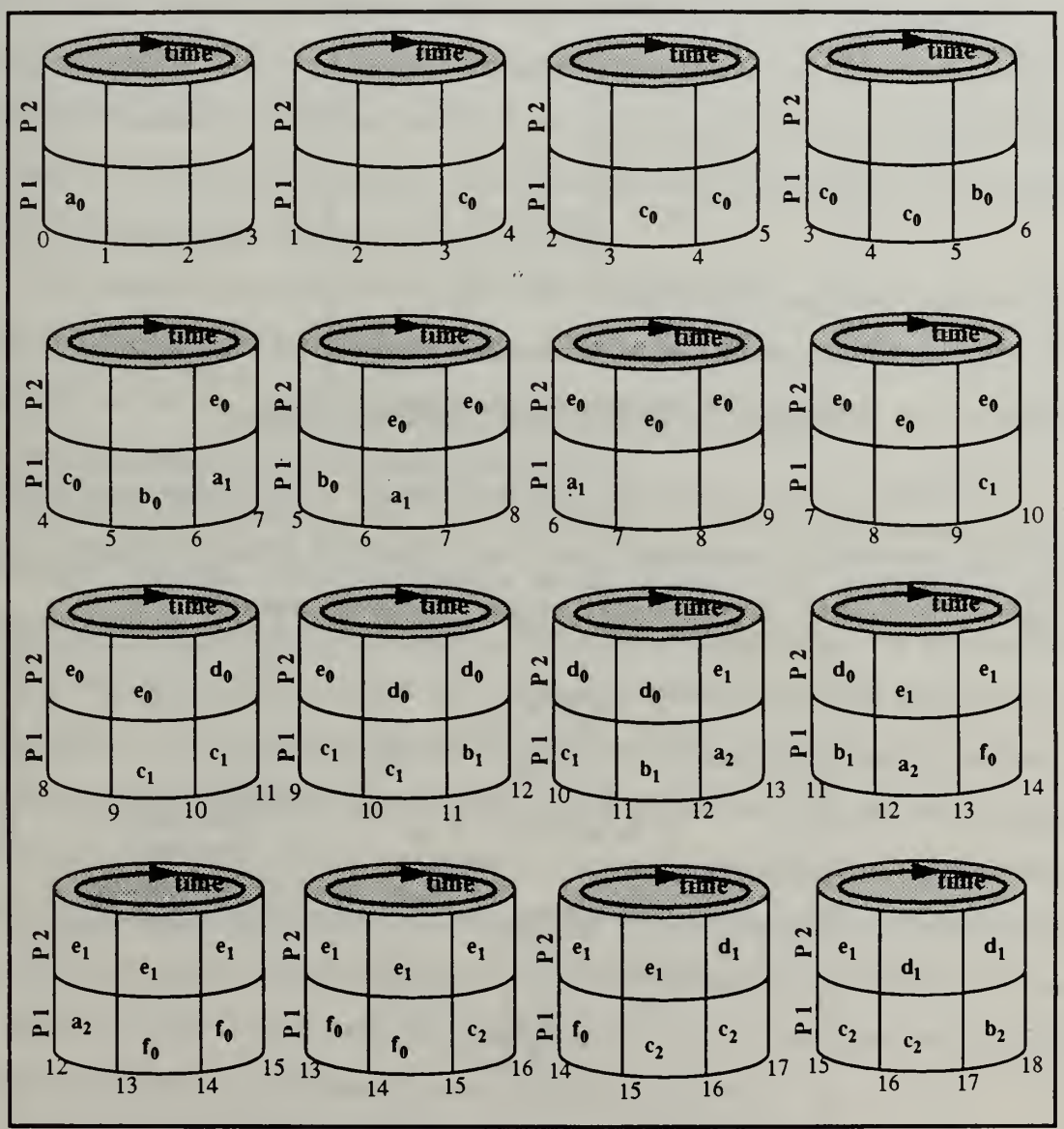


Figure 2.4. Execution of DAG I with Schedule I

B. IMPLEMENTATION OF RC

Once all nodes have been inserted into the cylinder, then node indices based on their location on the cylinder can be determined. Figure 2.5 depicts the node indices of DAG I on schedule I.

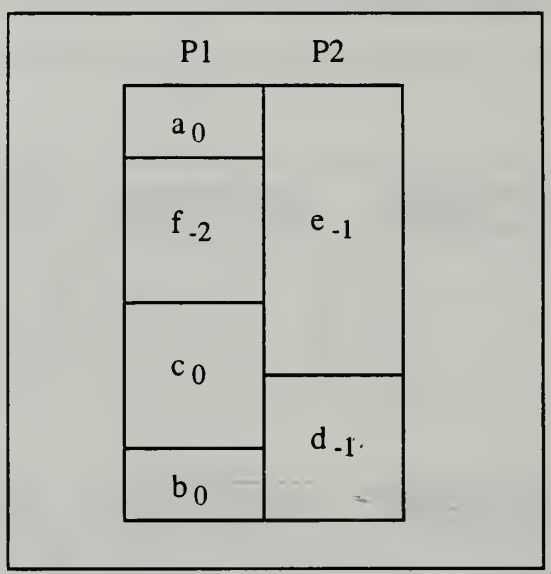


Figure 2.5. Node indeces of DAG I with schedule I.

Dependency arcs are created by using these indices. The dependency arcs belonging to DAG I according to the schedule I are shown in Figure 2.6. Detailed explanation of node index assignment, and dependency arc creation is given in Chapter IV.

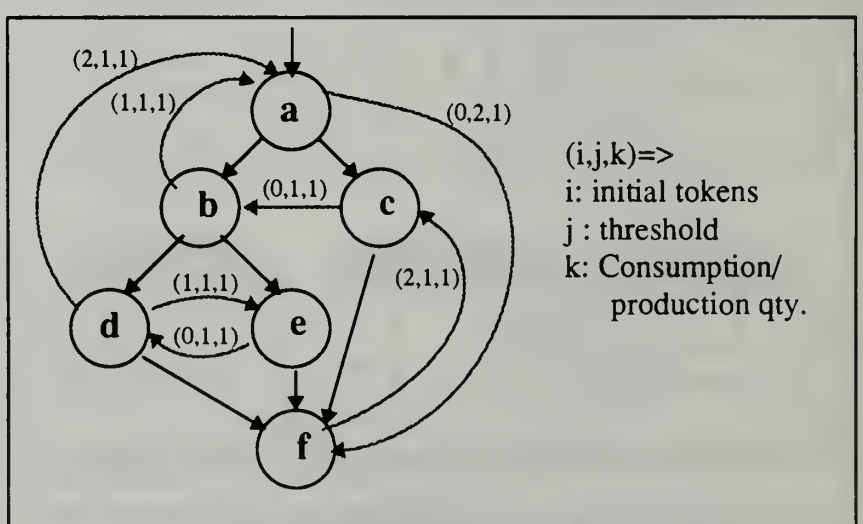


Figure 2.6. A Possible Restructure of DAG I

The labels on each dependency arcs indicates the initial tokens, threshold and consumed amount of sink node respectively.

C. POTENTIALS OF THE RC ANALYSIS

The RC technique of restructuring the application provides an improvement that the dependencies enforce node execution in order to provide more throughput and predictable response time. Without any extraneous control the FCFS scheduling can provide uniform throughput. The nodes receiving external data are ready for execution independent of the status of other nodes in the graph. If external data arrives more frequently than the execution frequencies of the lower nodes of the graph, they fall behind the upper nodes of the graph. This results in the upper nodes output queues going overcapacity, preventing them from entering the ready node list, [POPS 90].

It is possible to enforce the execution order loosely. Enforcing the cylinder loosely would make the system ***fully dynamic*** where the nodes are scheduled at run time only. It is preferable to run the system in ***fully dynamic*** mode. The RC technique and subsequent restructuring simply enhance the fully dynamic mode.

It is also possible to enforce the execution order strictly which would make the system ***fully static***. For each node of the graph a specific processor and exact time to begin processing can be given from the cylinder. This can be enforced directly by the scheduler to yield the ***fully static*** mode. But the running the system in the ***fully static*** mode is unwarranted. This mode of operation can be limited to the machines which do not have a dedicated run-time scheduler. The failure of a single processor will crash the whole system. This is unacceptable in a real-time system.Also since, all processors are assumed to be identical, it becomes unnecessary to assign a specific pocessors to a specific node. If a node is ready for execution, it can be assigned to the first available processor. This will reduce the amount of time a node waits for a processor in the fully static case, and provides flexibility and optimal utilization of system resources, [LEE 90].

III. SIMULATOR

A. PROGRAM MODEL

The input to Periodic Input Processing DAta Flow Simulator (PIPDAFS) is a directed graph. Nodes are the computation to be performed on the input data.

Data passing links between the nodes are FIFO queues. Each node reads data from its input queue, performs a nontrivial computation and writes the produced data to the output queues. A node is assured to carry no history. An example of a program graph which PIPDAFS will simulate is shown in Figure 3.1.

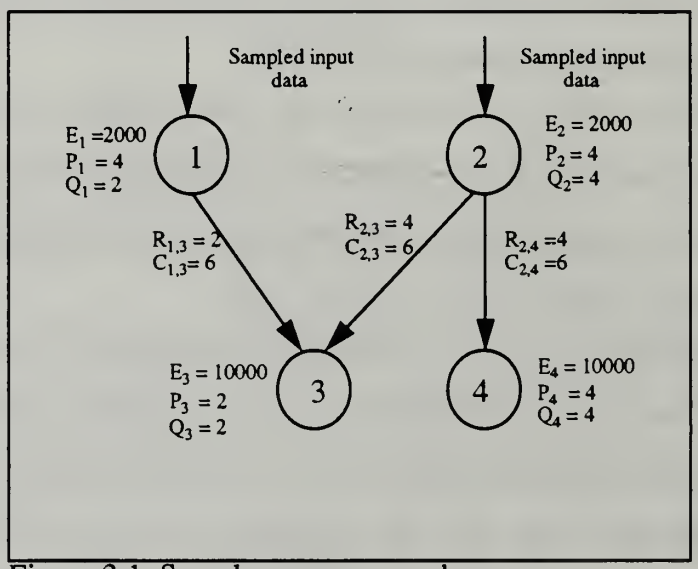


Figure 3.1. Sample program graph

The functionalities of the nodes are not simulated in PIPDAFS, only the resource requirements and computation times are. For each node and queue following quantities are prespecified and is fixed across different invocation of the graph.

Node Execution Time (E_i): It is the execution time of node i excluding any communication and synchronization overhead.

Data Production Amount (P_i): It is the amount of data that node i produces to its output queues for every invocation.

Data Consumption Amount (Q_i): It is the amount data that node i consumes from its input queues for each reading.

Data Threshold Amount ($R_{i,j}$): It is the least amount of data to be present in queue (i,j), so that node j can start execution.

Data Capacity Amount ($C_{i,j}$): It is the maximum amount of data queue (i,j) can store.

When a source node finishes its execution, it starts to write results to the output queues, output queues current lengths are incremented by production amount of source node. If current length of the queue is greater than threshold quantity then queue is said to be over_threshold. Queues can not be overcapacity since source nodes which will cause overcapacity can not be ready. Input queue sizes are updated as soon as a node consume its data from that queue. When data is consumed the queue current lengths are decreased by data consumption quantity. Input data for a node are queued and consumed in FIFO order. Node production amount can be different from the consumption amount as it is depicted in Figure 3.1. Queues do not only communicate data, but also, they can communicate synchronization information.

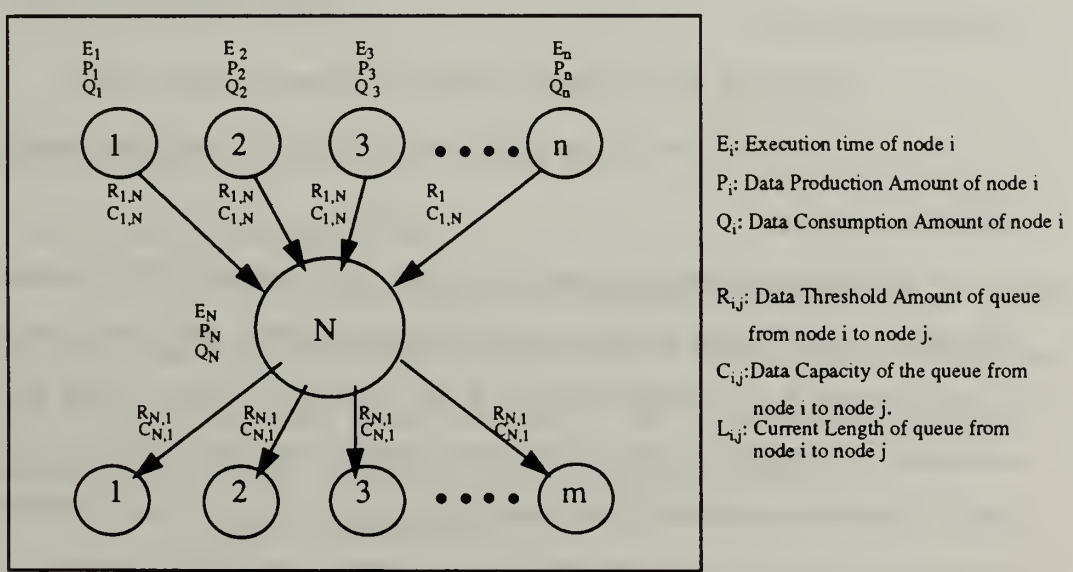


Figure 3.2. A sample graph node with its associated queues.

Assuming the node N in Figure 3.2 whose input queues are numbered from 1 to n and output queues are numbered from 1 to m.Let us accurately describe the conditions in which a node can be ready to be executed:

Queue Current Length ($L_{i,j}$): Current length of the queue (i,j).

Queue which links node i to node N, is over_threshold when;

$L_{i,N} \geq R_{i,N}$ for all i, $1 \rightarrow n$

Queue which links node i to node N, is overcapacity when;

$L_{i,N} > C_{i,N}$ for all i, $1 \rightarrow n$

When node setup is completed each input queue of the node is consumed.

$L_{i,N} = L_{i,N} - Q_N$ for all i, $1 \rightarrow n$

When a node is executed each output queue of the node is written.

$L_{N,j} = L_{N,j} + P_i$ for all j, $1 \rightarrow m$

A node is ready to execution when following conditions are satisfied.

i.) $L_{i,N} \geq R_{i,N}$ for all i, $1 \rightarrow n$ (All input queues are over_threshold.)

ii.) $L_{N,j} + P_N \leq C_{N,j}$ for all j, $1 \rightarrow m$ (All output queues has enough space to store the results.)

iii.) If t^n is the completion time of n^{th} instance of node i, then

$t^{n+1} > t^n + E_i$ for all i, $1 \rightarrow n$ (A node can not have multiple instances executing at a time.)

Input data arrival has a periodic nature. In every period I/O processors execute I/O nodes. The I/O nodes read the raw data from the external units such as sensors, format it, and forward it to its output queues. If the data arrival rate is higher than I/O node execution, then there is a chance of having new data overwrite the old data causing a data loss. It is the responsibility of application programmer, to ensure the program correctness by either choosing a computation which is not sensitive to mild data loss or by choosing an appropriate rate that can be met almost always by different machine components.

B. MACHINE MODEL

The machine model we assumed is based on three functional components: the Scheduler(SCH), Processors (GPs, IOPs), Global Memory Modules (GMMs). As data for the nodes becomes available, each node must be scheduled to execute on a processor. The machine model can be seen in Figure 3.3, and its functional components are described below in details.

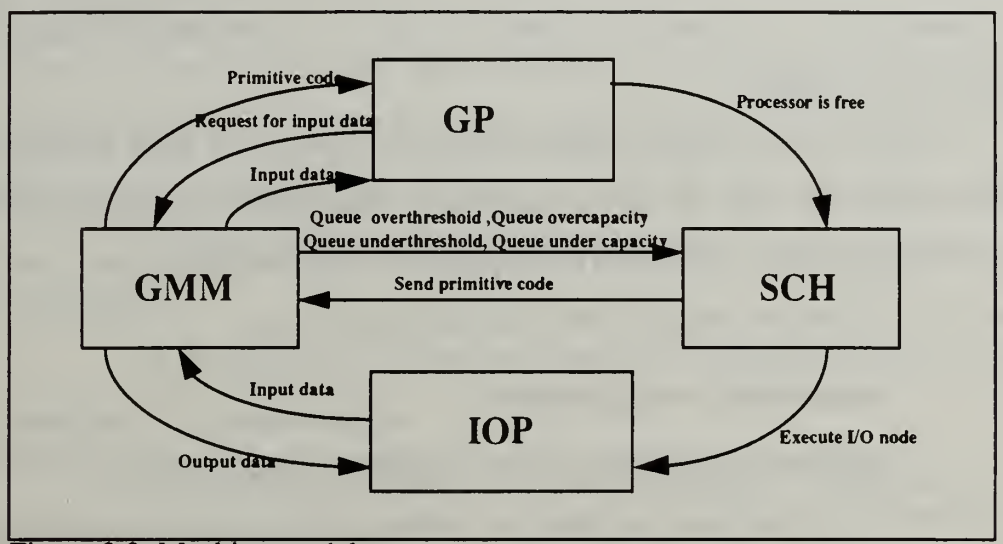


Figure 3.3. Machine model.

1. Processors

a. Generic Processors (GPs)

Based on the schedule signal a GP will read the primitive code and the necessary queue data from the GMMs (set-up stage), it will execute the designated primitives, and write the output queue data back to the GMMs.(breakdown stage). A maximum of three nodes can be associated with an GP at any one time

i.) One node being setup.

ii.) One node executing.

iii.) Another node being broken down.

It is important to note that, if a node is being broken down, a node assigned to set up on the GP can not start to set up (although it has been assigned to that GP).

Allowing set up and break down to overlap with execution is the main mechanism for allowing computation-communication overlap.

A processor is said to be free and available for reassigning by the scheduler, if the setup stage is not busy. Needless to say that this concept of a free processor is different from the classical definition of what a free processor is. A processor can still be executing and be considered free. The purpose of that is to allow for maximum computation and communication overlap.

When the data is ready for an internal node, SCH sends a signal to GMM to send code to assigned GP. Each GP has a local memory which capable of storing the primitive code and the input data. As soon as GP acquires the code seeks input data from GMMs. When a GP finishes node set up informs the SCH that it is free.

b. Input/Output Processors(IOPs)

Input nodes periodically receive data from the sensors. IOPs execute the input node of DFG by formatting the raw data and writing to their output queues. Output nodes are also executed at IOPs and IOPs redirect the ready data to the next stage in the system.

When the data is ready for input or output nodes, SCH sends a signal to an IOP to execute the specified input or output node. IOPs send a signal to GMMs for writing/ reading.

2. Global Memory Modules(GMMs)

The GMMs provide the data storage for the machine. These GMMs are different from a typical computer memory, since each is proactive (each has some sort of a processor) and operates independently. Each data queue is allocated to a single GMM for storage. When a GP starts node setup, input data queues are consumed. When a GP finishes execution all the output queues from the completed node are produced (written) to the appropriate GMMs. After each produce and consume queue, current lengths are updated.

GMMs check for over_threshold or overcapacity conditions. If GMM recognize a over_ threshold informs the scheduler. When a node is assigned to an GP, SCH instructs to GMM to write primitive code to the GP's local memory. Upon receiving the primitive code by GP, it asks appropriate GMMs for input data. GMMs only communicate with one processor at a time. If there are requests while GMM is busy, these requests are waited for GMM to be available.

3. Scheduler

In this model, SCH works like a dispatcher and maintains two list one for the ready nodes and the other one for processors which indicates the free processors. SCH dispatches the ready nodes to the free processors. A node can be in three different state

i.) Ready node; which is waiting in the ready list to be assigned to a free GP to be executed.

ii.) Processing node; which is assigned to a processor and is in one of the setup, breakdown or execution stages.

iii.) Waiting Node; which is waiting for input data's being ready or output queues having enough space or both.

A node becomes ready, if all of its input queues are over threshold and output queues will not be over_ threshold after the node execution. When a node becomes ready, it is put into the ready node list and then SCH attempts to match a free processor to a ready node. When a processor finishes node setup and start to node execution, it is indicated as free processor, and SCH checks the ready node list, if ready node list is not empty then attempts to match a ready node to the free processor.

We assume that SCH runs on a dedicated processor.[1]

---

1. Alternatively Scheduler could be made to run on a CP with a high priority.

C. SIMULATOR DESCRIPTION

PIPDAFS is an event driven simulator, events are stored in a queue which is prioritized by their time stamps. The simulation terminates when a certain number of instances (prespecified by the user) has been executed. The event queue is first initialized with events denoting readiness of the input nodes, then the rest of the events are produced from these initial events. An instance is counted as started when one of the input nodes belonging to that instance has started to setup. An instance is counted as completed, when all of the output nodes belonging to that instance have finished breakdown. The events that may occur during the simulation are shown in Figure 3.4. In Figure 3.4 simulation starts from the reach_producton_event and terminates with finish_breakdown event.The events are;

*1) **Reach_production_period***: This event is produced periodically by the external input data and if the output queues of input node has enough space, makes the input nodes ready to execute. This event produces the ***Inputqueuesoverthreshold*** and the ***Reach_production_period*** events.

2) ***Inputqueuesoverthreshold***: This event indicates that all input queues of a node are over_threshold. In other words, input data is available for node execution. While processing this event output queues are checked whether they enough space to store the results or node is an output node. If the output queues have space or the node is an output node then the ***Ready_node*** event is produced to indicate that node is ready to be executed.

*3) **Ready_node***: This event indicates that node is ready to be executed and can be put into the ready node list and be scheduled to a free processor. If a previous instance of the ready node is currently executing then according to the policy (defined by the user before simulation) ready node may be put into the ready node list, but not executed until previous instance has been completed or ready node not even put into the ready node list. For each ready node added the ***Schedule_a_node_from_ready_list*** event is produced.

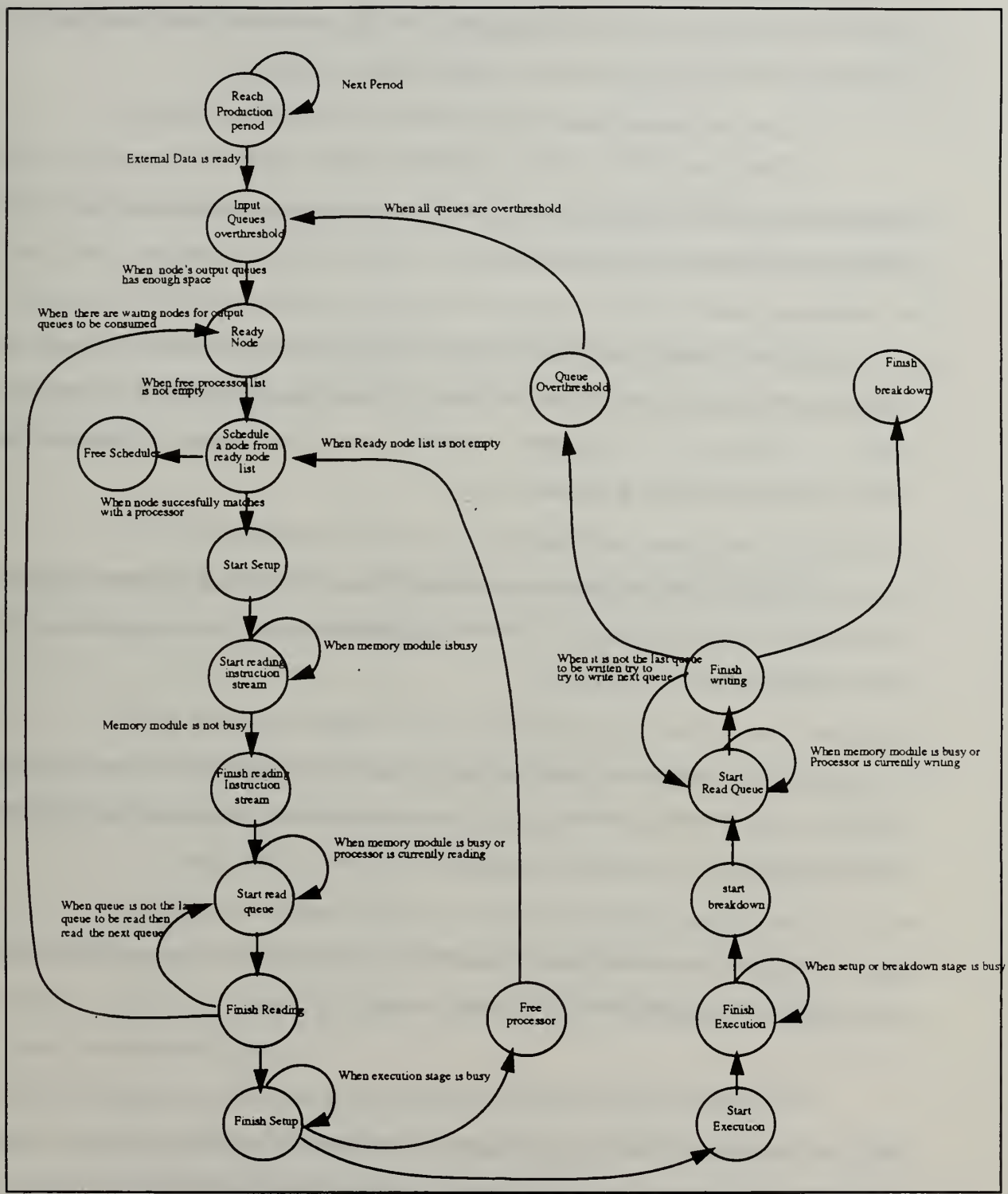


Figure 3.4. Events that are produced during the simulation

4) Schedule_a_node_from_ready_list: If both the free processor list and the ready node list were not empty. This event attempts to match a ready node to a free processor. If the attempt is successful, it produces ***Start_setup*** event.

5) Start_setup: This event is produced, when a ready node matches to free processor successfully. If the node is the first input node of a new instance, then it will indicate the time for a new instance start.

6) ***Start_reading_instruction_stream***: This event keeps trying to read the instruction stream until the memory module, where instructions are stored is not busy. Then it marks the memory module as busy and produces the ***Finish_reading_instruction_stream*** event.

7) ***Finish_reading_instruction_stream***:This event marks a memory module as not busy and for each input queues of the node whose instruction stream was accessed, the ***Start_readqueue*** event is produced.

8) Start_readqueue: This event keeps checking a memory module and assigned processor until memory module is not busy and processor is not reading, then the ***Finish_reading*** event is produced. Reading is simulated by time delay.

9) Finish_reading: This event completes the reading of the queue. If it is the last queue which is to be read, then ***Finish_setup*** event is produced.

10) Finish_setup: If processor is not executing and breakdown stage is not busy, setup can be finished. When setup is finished, ***Free processor*** and ***Start_execution*** events are produced. Otherwise finish_setup event is reentered with a new time stamp.

*11) **Free_processor***: This event indicates that processor is free and produces the ***Schedule_a_node_from_ready_list*** event is produced.

*12) **Start_execution***:This event produces ***Finish_execution*** event.

*13) **Finish_execution***: If breakdown and setup stages are not busy, then produces the ***Start_breakdown*** event else checks whether setup stage is waiting for execution to finish or not. When setup stage is waiting for execution to finish, ***Finish_execution*** event is produced in order to prevent deadlock.If neither setup stage is waiting for execution stage nor setup and breakdown stages were not ready then ***Finish_execution*** event reentered with a new time stamp.

*14) **start_breakdown***: This event produces ***Start_write_queue*** event for each output queue and indicate the assigned processor as not free. Since setup and breakdown can not happen concurrently in this model.

*15) **Start_write_queue***: If memory module to be written is not currently busy and processor is not currently writing, then the ***Finish_writing*** event is produced else ***Start_write_queue*** is reentered with a new time stamp.

*16) **Finish_writing***: This event completes the writing to the queue. If it is the last queue to be written, produces the ***Finish_breakdown*** event. Updates the output queue's current length. If queue length is over_threshold produce the ***Queue_overthreshold*** event.

*17) **Finish_breakdown***:This event completes the breakdown. If the node completed is the last output node of that instance, it is assumed that an instance is finished and instance finish time is written a file namely endtimes.

18) Queues_overthreshold: This event checks the all input queues whether all are over_threshold or not. If all are over_threshold, then the ***Inputqueuesoverthreshold*** event is produced.

19) Free scheduler: This event marks the scheduler as not busy and produces *Schedule_a_node_from_ready_list* event. Scheduling time can be defined by the user. This is useful in runs when the overhead of scheduling is a problem research.

D. SIMULATOR PROGRAM DATA STRUCTURES

The simulator is written in C++. Its data structure and class dependencies are depicted in Figure 3.5. The simulator code is given in Appendix B.

E. USER INTERFACE

1. Input

To execute the simulator user simply invokes the executable version and follows the on-screen prompts. The simulator collects data about machine from file ***machine.config***. Graph data is stored in the file ***graph.dat***. Also the memory assignments is given in the file ***memory.dat***. A sample graph data and a machine configuration file is given in Appendix A.

Besides the graph and the machine files, user is prompted for the following data:

- ***Instance Start Number***: The instance number when information gathering is started.

- ***Instance End Number***: The instance number when the simulation will be completed.

- ***Instance Overlap***: It the amount of overlap between the multiple instances of the same node. It could be none which means while a node instance is ready, another instance of the same node can not be ready. Or it could be overlapped which means multiple instances of a node can be ready in the ready node list, but only one of them executed at a time.

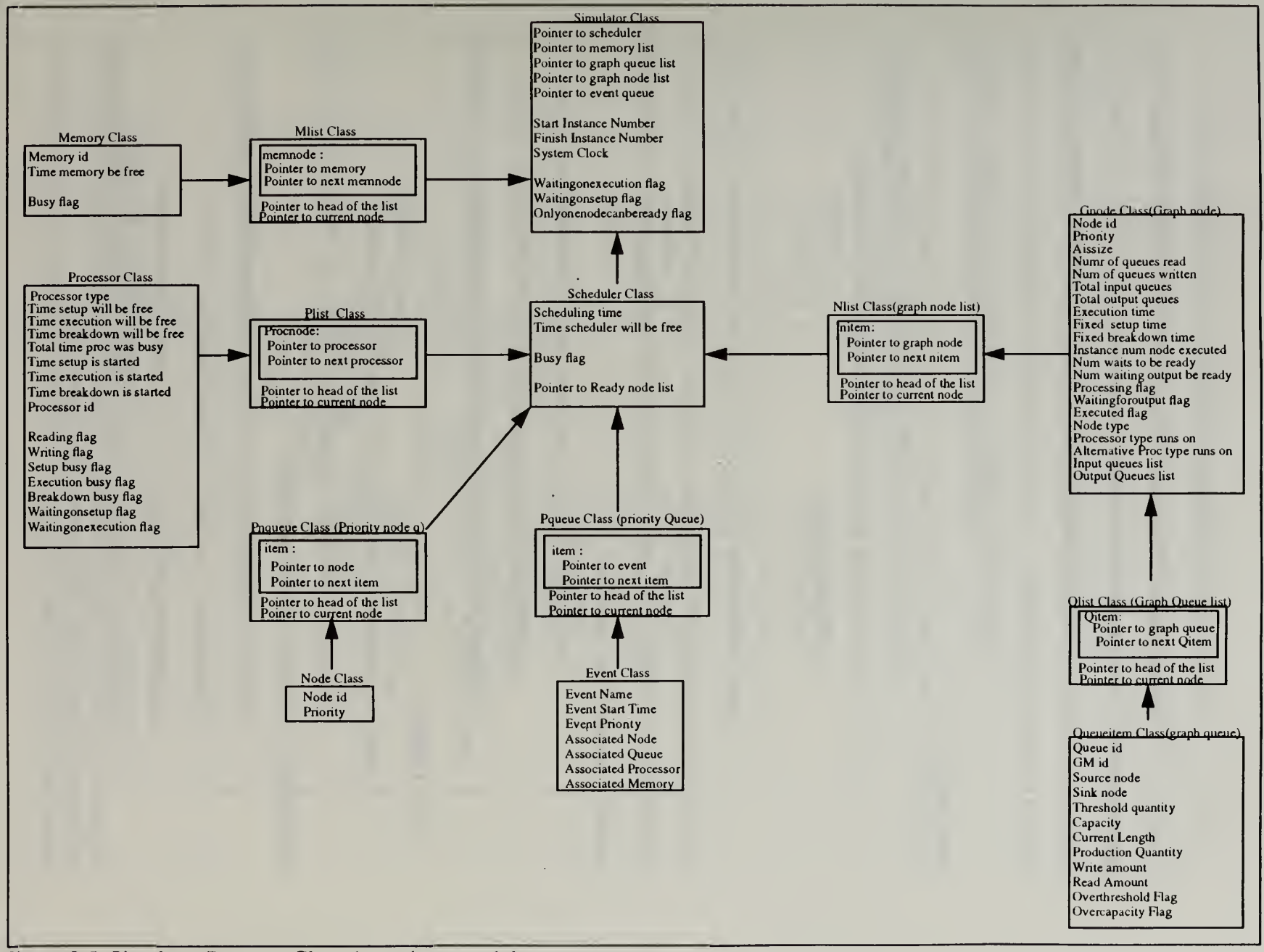


Figure 3.5. Simulator Program Class dependency and data structure.

- ***Node Execution Priority***: It is the priority for scheduling the nodes in the ready node list. User choose of the following:

Shortest First (Execution time): Nodes in the ready node list are ordered by their execution times.

Longest First (Length): Nodes in the ready node list are ordered by their length.

Random: Node's priorities are defined by the simulator randomly.

User Defined: Node's priorities are defined by the user.

No Priority(FCFS)

- ***Data Period***: User may want to repeat the simulation for different data periods without changing the other data. This time in order to prevent, the user from reinvoking the simulator. The initial data period and final data period with increment amount is asked by the simulator.

Start Period: It is the data period where the simulation will start.

End Period: It is the data period where the simulation will finish.

Interval: It is amount of increase in the data rate for after each simulation.

2. Output

After the simulation is completed, following data is produced:

- ***Processor Utilization*** (Considering setup and breakdown as useful utilization): It is stored in file ***utilization.dat.***

- ***Processor Utilization*** (Only execution): It is stored in file ***execution.dat.***

- ***Throughput***: It is stored in file ***throughput.dat***

- ***Response Time***: It is stored in file ***resptime.dat***

- ***Coefficient of Variation***: It is ratio of instance length standard deviation to the average instance length. It is stored in file ***Coefvar.dat***

Files also contain corresponding normalized data rate values.

If the user prefer to store the events occurring during simulation, a log file can be produced. Log file includes ***time***, ***event name***, ***node number***, ***queue number***, ***memory module***, ***processor number***. Any user who interested in different kinds of statistics can obtain desired information by filtering the log file.

F. EXAMPLE RUNS OF THE SIMULATOR

The graph depicted in Figure 5.1 is executed with simulator. The graph belongs to a real application which is called correlator.

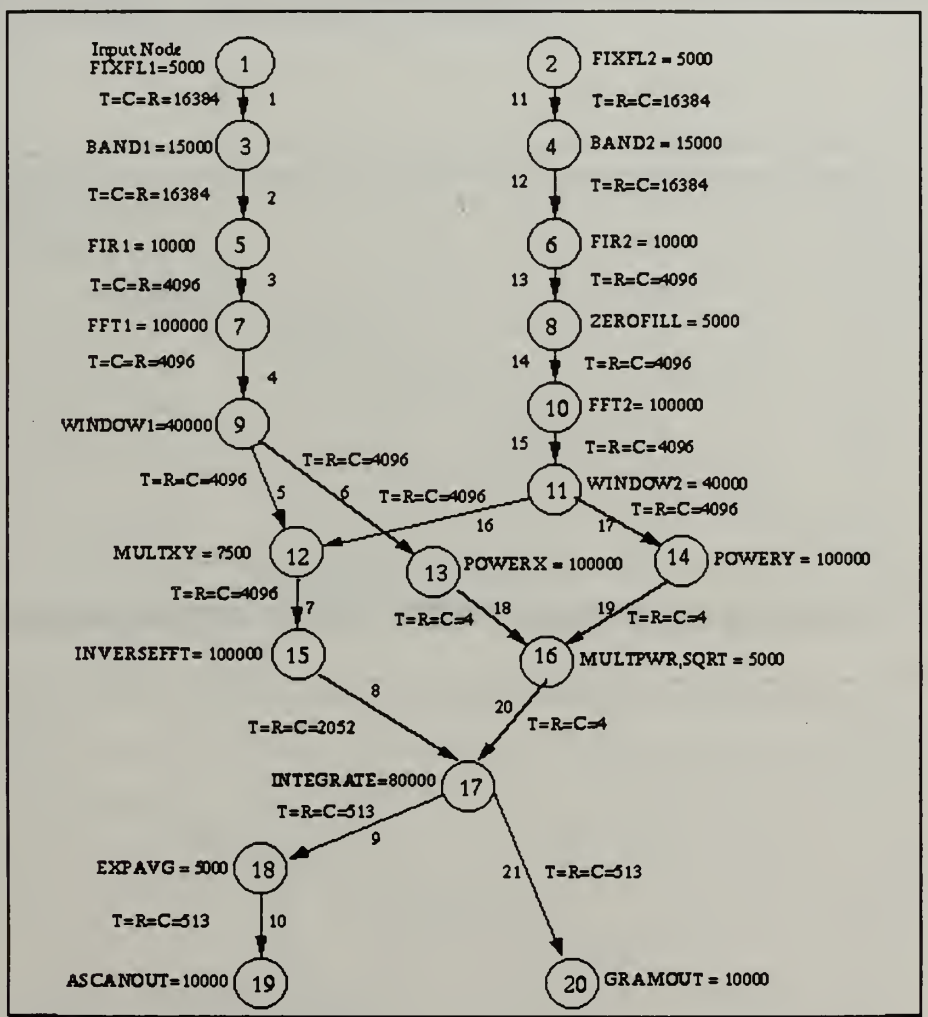


Figure 3.6. Correlator graph

The results belonging to correlator graph are shown in Figure 3.7, Figure 3.8, Figure 3.9. In resultant graph, X-coordinate refers to the normalized data rate which is the ratio of

maximum throughput rate to the input data rate. Maximum throughput rate means total execution time over number of processors.Y-coordinate refers to throughput, utilization or coefficient of standard deviation. Throughput is the number instances that are executed in unit time in this simulator which one second. Utilization shows processor usage percent, obtained by dividing processor busy time to total execution time. Two kinds of utilization is calculated for processors, one consider the setup and breakdown as useful utilization and the other one does not. For each data rate 500 instance were executed and statistical result are obtained from last 300 hundred instances.

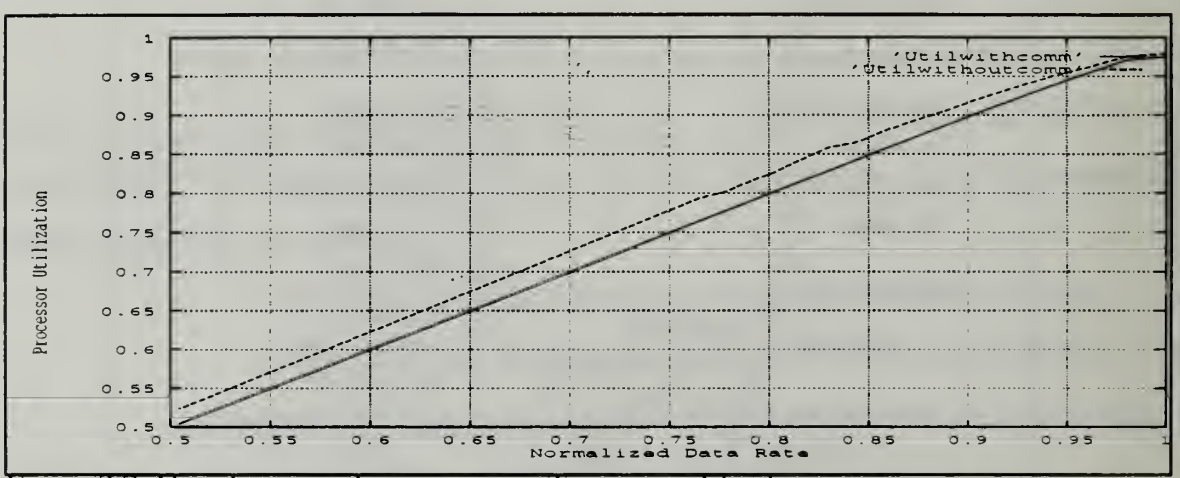


Figure 3.7. Correlator graph, processor utilization with/without communication.

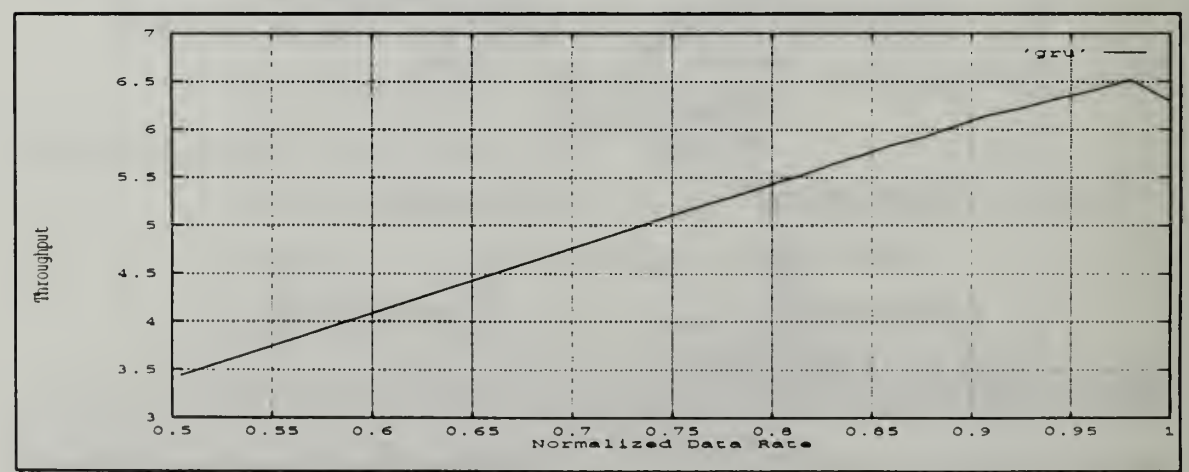


Figure 3.8. Correlator graph, throughput.

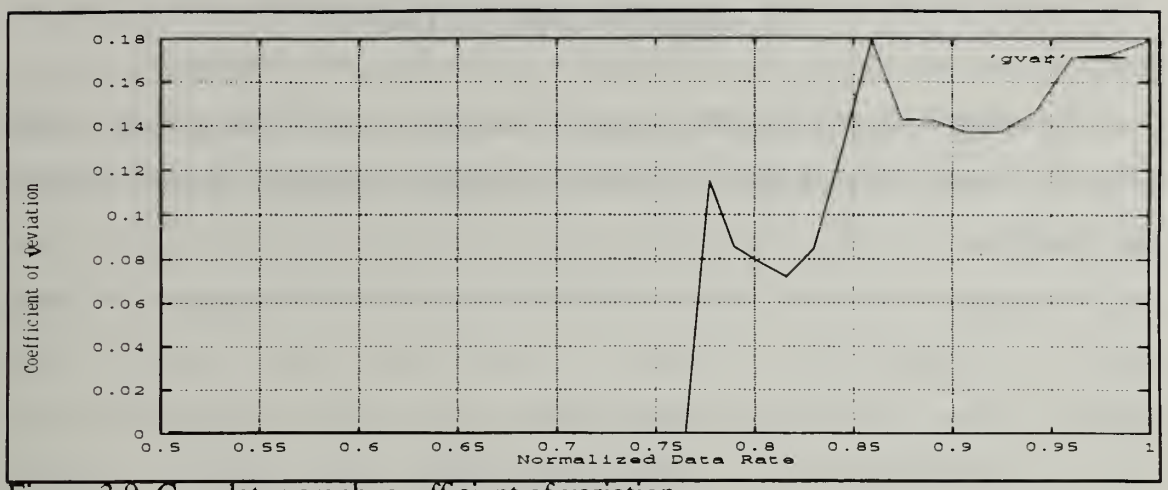


Figure 3.9. Correlator graph, coefficient of variation

IV. GRAPH RESTRUCTURING

This chapter describes the graph restructuring that enforces RC scheduling in detail. Also, it evaluates the potential of RC-based scheduling techniques through a series of experiments.

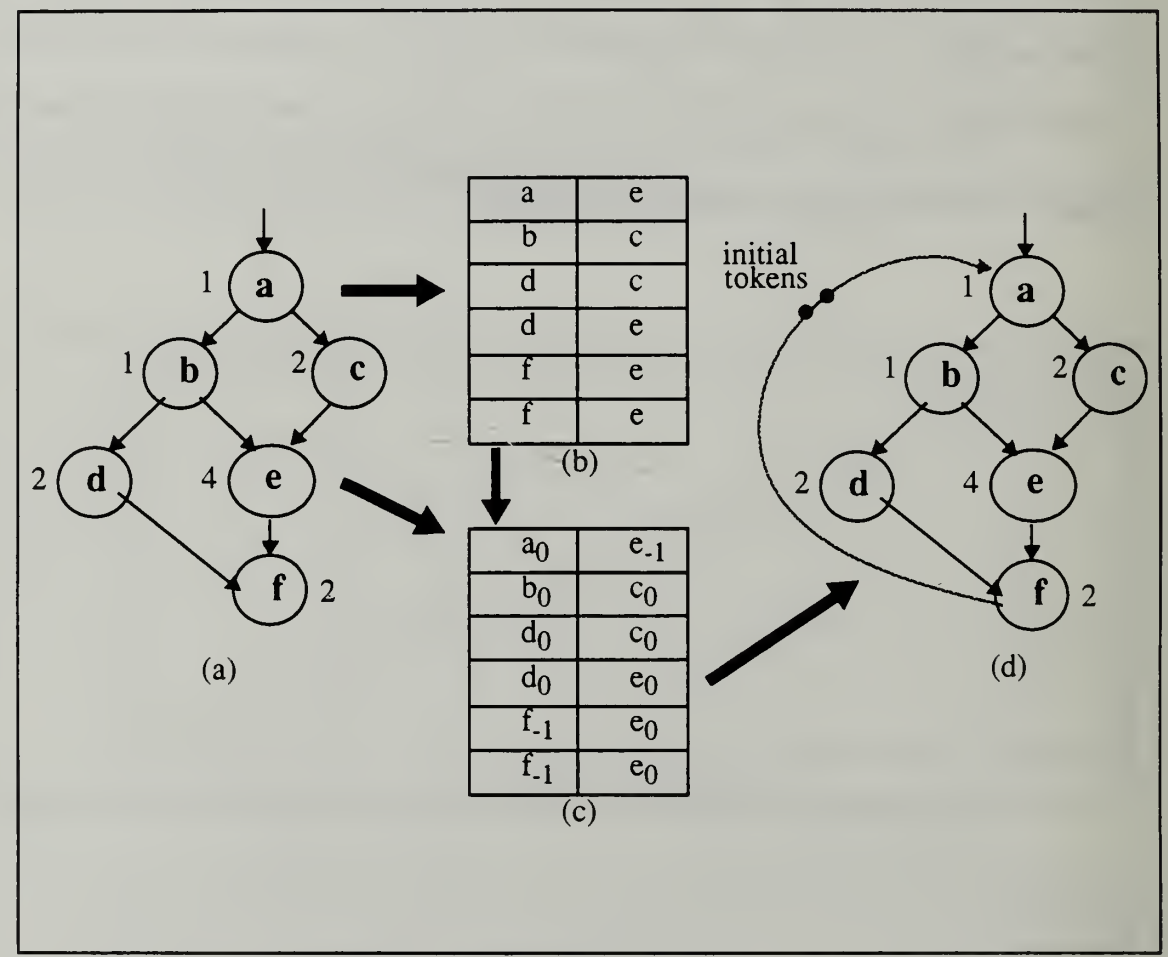


Figure 4.1 An example for graph restructuring.

Figure 4.1 describes the steps that are needed to restructure an application graph and make it ready for execution.The first step in the process is ***cylinder mapping***. At this point, graph nodes are mapped to the cylinder according to a given objective.[1] The circumference

1. In the Figure 4.1b the mapping is based on fitting the graph by using the nodes in a topologically sorted list.

of the cylinder is determined by the sum of the nodes' execution times divided by the number of processors (an approximation which does not consider the communication delay). The mapping shown in Figure 4.1(b) assumes two processors. After the cylinder is mapped, the next step is ***index assignment***. In this step, nodes are given relative indices that clarify which node is executing which relative instance. For example in Figure 4.1(c), while ***node f*** is executing, ***node e*** can execute one later instance. Indices are determined by the nodes' locations on the cylinder.Figure 4.1(c) depicts the index assignment. After the indices are assigned, the next step is ***synchronization arcs creation*** to enforce the given schedule in Figure 4.1(c). Figure 4.1(d) shows the synchronization arcs. The synchronization arcs that are obtained are pruned by removing the redundant arcs. And the resultant arcs are added to the original graph. The restructured graph with the added synchronization arcs is depicted in Figure 4.1(d). The program that restructures the graph is called ***Graph Restructurer*** (GR) and its organization is represented in Figure 4.2.The source code of GR is given in Appendix C.

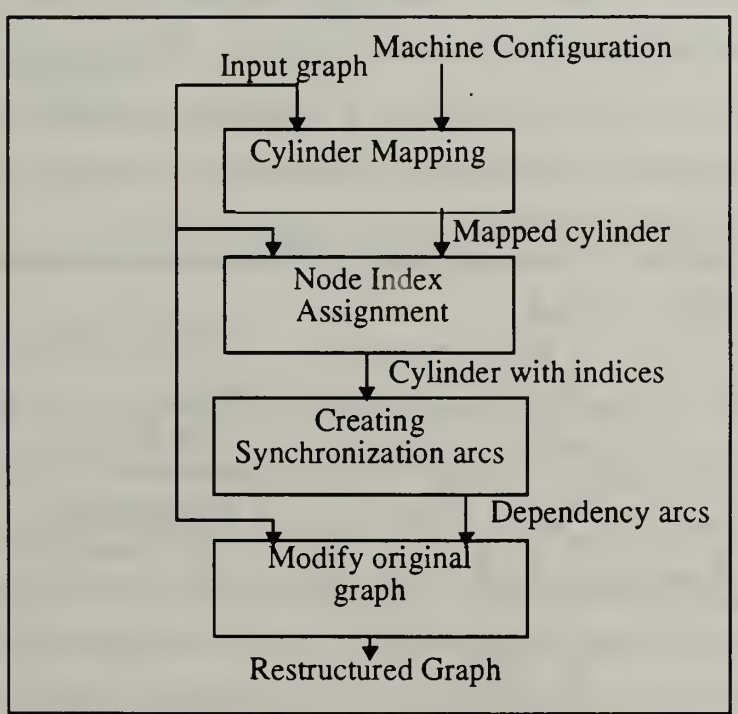


Figure 4.2. The structure of the graph restructurer.

The GR basically has three functions, cylinder mapping, index assignment and the synchronization arcs creation.

A. CYLINDER MAPPING

The cylinder mapping component of the GR takes the original graph and machine configuration as input data. In [LIT 91] and [BELL 92], nodes are ordered for mapping according to their execution times. The graph topology is not considered. This produces more dependency arcs increasing the communication overhead. Figure 4.3 represents the dependency arcs for the same graph with a random mapping. This random mapping produces 7 synchronization arcs, but the mapping which considers the graph topology produces only one synchronization arc. In Figure 4.4 the mapping algorithm which considers the graph topology is given.

In the given algorithm, first the nodes are sorted topologically, then mapping starts from the root node and continues down the sorted list of the nodes. The mapping heuristic is to map a child node to an empty space, such that it can start after its parent's finish time. If no such space can be found, then the node is placed in the earliest start empty space (probably resulting in a synchronization arc). If no attempt to find any empty space is successful, then the cylinder circumference is incremented by a percentage specified by the user, and the mapping process is restarted all over again.

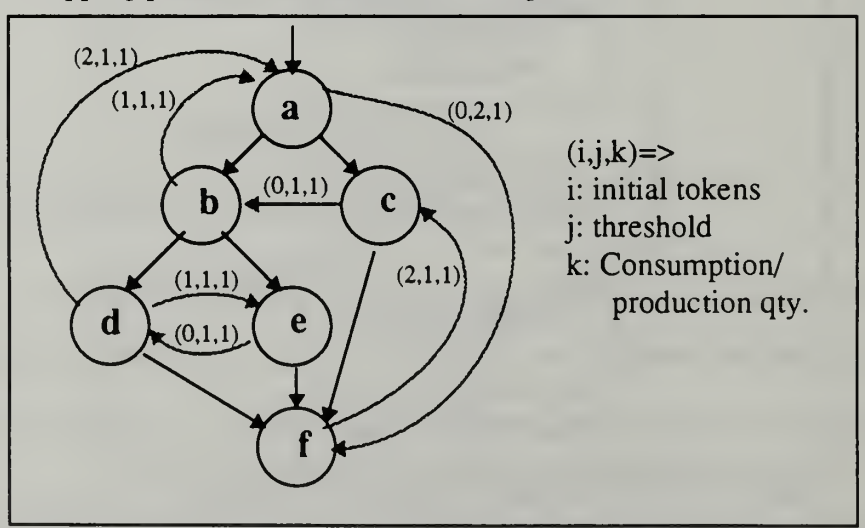


Figure 4.3. Dependency arcs with random cylinder mapping.

```
procedure Map_the_cylinder(G,C) /*G is Directed Acyclic Graph*
                                /* C is the cylinder*/

q ← topological_sort(G); /* q is a queue*/
boolean Success= true

for each node sorted topologically in the graph
    Try to place the node to an empty space starts after latest end
parent node;
   if there is no empty space
      try to place the node to the empty space that starts before the
latest ends parent node, but has enough space to place the node
after the latest parent node;
   if there is no empty space
     try to place the node to the earliest available empty space.
  else if there is no empty place
          success = false;
          increment the circumference of the cylinder;
          Map_the_cylinder(G,C);
          exit;
end (for);
end Map_the_cylinder.
```

Figure 4.4. Algorithm for cylinder mapping.

B. INDEX ASSIGNMENT

The data flow execution of the graph requires the analysis of the assignment of the nodes to the cylinder to determine which instance of a node is actually to be executed. The algorithm for index assignment is given in [BELL 92].

In this algorithm, an arbitrary node is assigned an index of zero to represent the fact that it is the current instance of this node that is being executed. Every parent and child of this node is examined with respect to its relative position on the cylinder.

For a node N, Indices of the parents and children nodes are determined according to the following conditions:

1. If a parent's completion time on the cylinder is greater than the node N's start time on the cylinder, then the parent must be executing a later instance.Therefore, the parent is given an index which is incremented by one.[2]

2. If a parent's completion time on the cylinder is less than or equal to the node N's start time on the cylinder, then the parent can be executing the same instance with the current node. Therefore, parent is given the same index as node N.

3.If a child's start time on the cylinder is less than the node N's finish time on the cylinder then the child must be executing a previous instance. Therefore, the child is given an index which is decremented by one.[3]

4. If a child's start time on the cylinder is greater than or equal to the node N' finish time on the cylinder then the child can be executing the same instance with the node N. Therefore, the child is given the same index as the node N.

The original graph and the cylinder mapping is given to the algorithm as input and the node index assignments are output to a file.

C. SYNCHRONIZATION ARCS CREATION

After the cylinder assignment and corresponding node indices are established. The required synchronization arcs for a given RC schedule can be determined.

A synchronization arc is a logical arc which enforces a given execution sequence. For example, in Figure 4.1(c), ***node b*** and ***node c*** can be executed after ***node a***'s completion, ***node d*** can be executed after ***node b***'s completion, ***node e*** can be executed after ***node c***'s completion, ***node f*** can be executed after ***node d*** and ***node e***'s completion and ***node a*** can be executed after ***node f***'s completion. The execution sequence of all nodes except ***node a***

---

2. Some value greater than one would be acceptable, although the parents output queue size will place an upper bound on that value.
3. Some value less than one would be acceptable, although the child input queue size will place an lower bound on that value.

is satisfied by the data dependency. To control the execution of node a, we need an arc from node f which will trigger ***node a*** after it completes its execution. This trigger is called synchronization arc. These synchronization arcs can be implemented as input queues or output queues.The synchronization arcs implemented as input queues are called logical synchronization arcs, and the synchronization arcs implemented as output queues are called physical synchronization arcs.

In order to understand what logical and physical synchronization arcs means, let us look the Figure 4.1 again. In this Figure, after restructuring the original graph we obtain an synchronization arc from ***node f*** to ***node a*** with two initial tokens. It simply prevents the ***node a***'s third execution until ***node f*** finishes its first execution. This arc can be implemented as an input token queue which has an initial number of tokens, a threshold, and a consumption and production amount. During the execution, these tokens are produced and consumed according to the sink and source node execution.

Enforcing the RC schedule with logical arcs is called Start After Finish (SAF) approach. It can be easily seen from the Figure 4.1e. ***node a*** can not start its third execution until ***node b*** finishes its first execution.

The same synchronization arc can also be enforced by putting a physical arc. This physical arc can be implemented as an output token queue. In this approach, execution sequence is controlled by the queue capacity instead of queue threshold quantity. Figure 4.5 depicts the physical synchronization arc and the logical synchronization arc which correspond to the same synchronization arc.

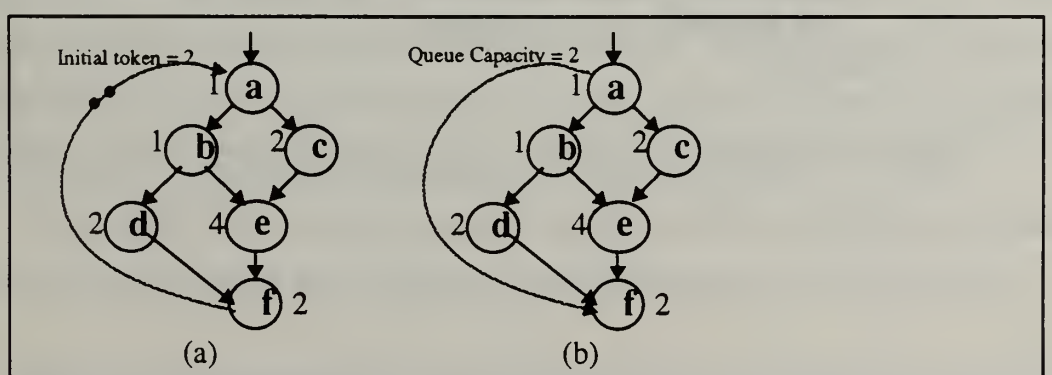


Figure 4.5.(a) A restructured DAG with logical arcs. (b) A Structured DAG with physical arcs.

Enforcing the RC schedule with physical arcs is called Start After Start (SAS) approach. It can be easily seen from Figure 4.4 (b) that ***node a*** can not start its third execution until ***node f*** starts its first execution.The algorithm for SAF and SAS approaches are given in Figure 4.6 and Figure 4.7 respectively.

```
procedure Restructure_graph_with_SAF (Cylinder);
  /*n_r,n_s are nodes of graph, G*/
  for all nodes, n_r
     check index i of n_r
     find the latest node, n_s that ends
          before n_r starts on the cylinder
     check index j of n_s

     /*if n_r starts at the top of the cylinder, the latest*/
     /*node ends at the bottom of the cylinder.*/
     /*In this case, j should be decremental by one*/

     introduce a synchronization arc from n_s to n_r
     if i ≥ j
       put i - j initial tokens on the arc
       set threshold = 1, consume = 1
     else if i < j
       put 0 initial tokens on the arc
       set threshold = j-i+1, consume = 1
  end (for).
```

Figure 4.6. Algorithm to generate logical synchronization arcs, [SLZ 92].

```
procedure Restructure_graph_with_SAS (Cylinder);
  /*n_r,n_s are nodes of graph, G*/
  for all nodes, n_r
     check index i of n_r
     find the latest node, n_s that starts
          before n_r starts on the cylinder
     check index j of n_s

     /*if n_r starts at the top of the cylinder, the latest*/
     /*node starts at the bottom of the cylinder.*/
     /*In this case, j should be decremented by one*/

     introduce a synchronization arc from n_s to n_r
     if i > j
       put i - j initial tokens on the arc
       set threshold = 1, consume = 1
     else if i < j
       put 0 initial tokens on the arc
       set capacity = j-i+1, consume = 1, threshold = 1
  end (for).
```

Figure 4.7. Algorithm to generate physical synchronization arcs.

D. PERFORMANCE EVALUATION

In our performance evaluation experiments, we use two sample graphs one is a graph which is used for active sonobuoys and the other is an artificial test graph with a controlled topology. The benchmark graph consists of 50 nodes and 139 data queues. It is depicted in Figure 4.8.The small test graph consists of 25 nodes and 27 data queues. It is depicted in Figure 4.9.

In the experiments, in order to minimize the effect of the communication overhead, the backward synhronization arcs from lower nodes to the upper nodes, are removed

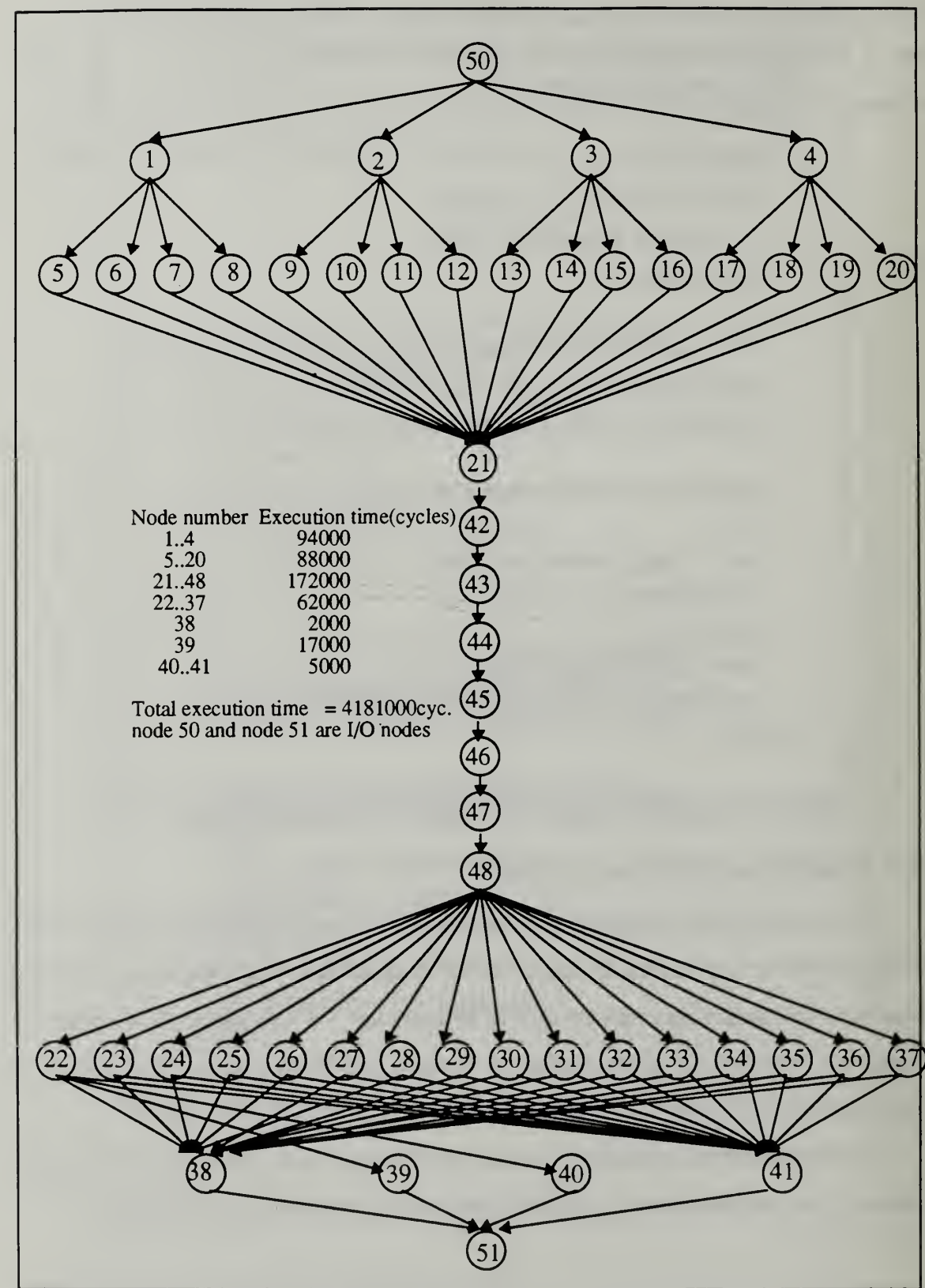


Figure 4.8. Active sonobuoy benchmark graph.

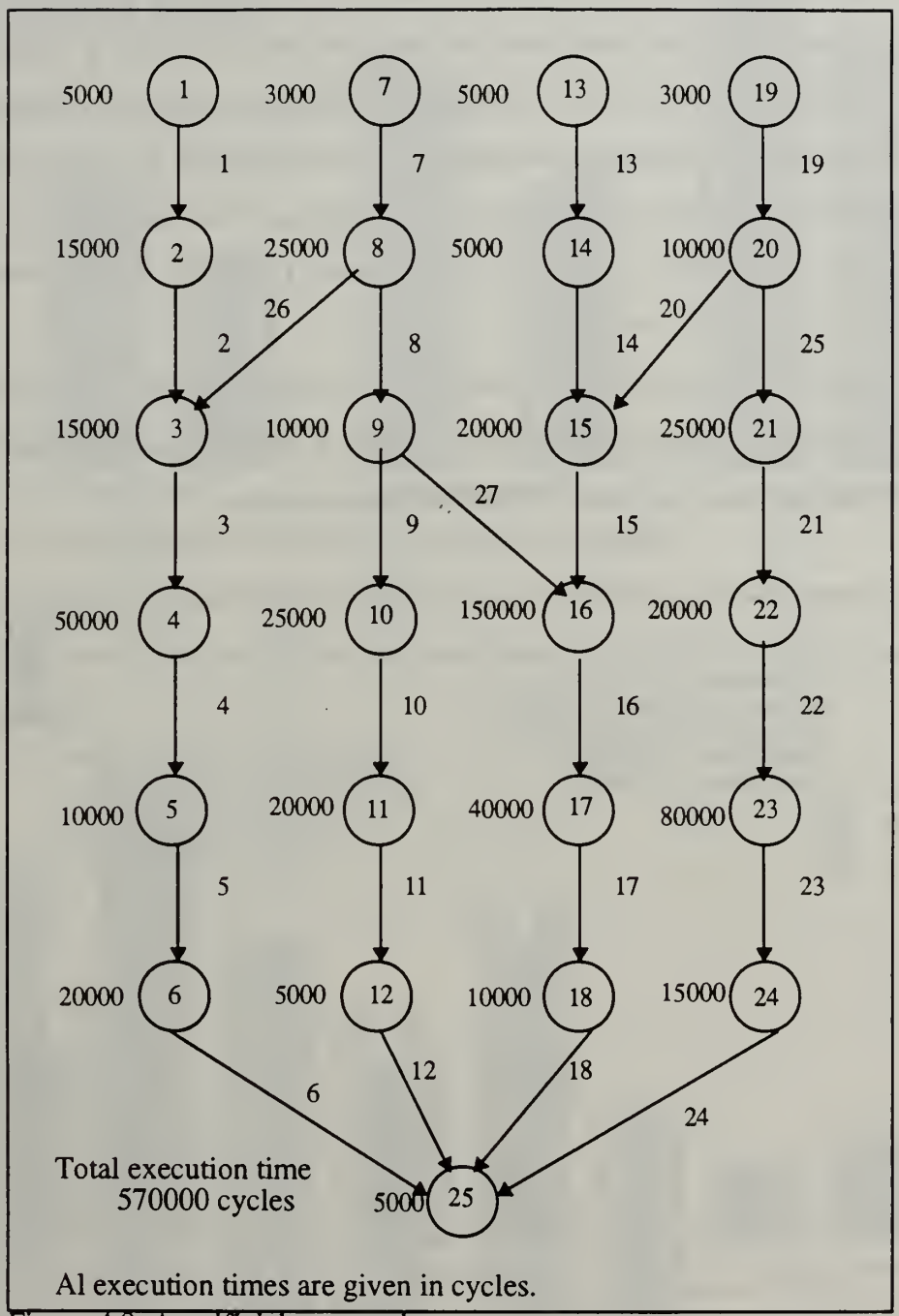


Figure 4.9. A artificial test graph.

1. Basic Performance Measurements

In this experiment, we examine the performance of SAS, SAF, and FCFS scheduling techniques running on different numbers of processors and arbitrary queue to memory mapping. The experiment is repeated twice, assuming high and low communication overheads. Low and high communication overheads correspond to 0.15 and 0.75 percent communication-computation ratio respectively. In order to chose a data rate, first FCFS scheduling is simulated and the data rate which saturates the machine is chosen. The cylinder then is mapped at this data rate. The results demonstrate that FCFS has a high throughput in most cases.

Figure 4.10 and figure 4.11 depict the throughput results from the benchmark graph and artificial test graph respectively. In Figure 4.10, SAF scheduling exhibits slightly better throughput than FCFS and SAS scheduling techniques.

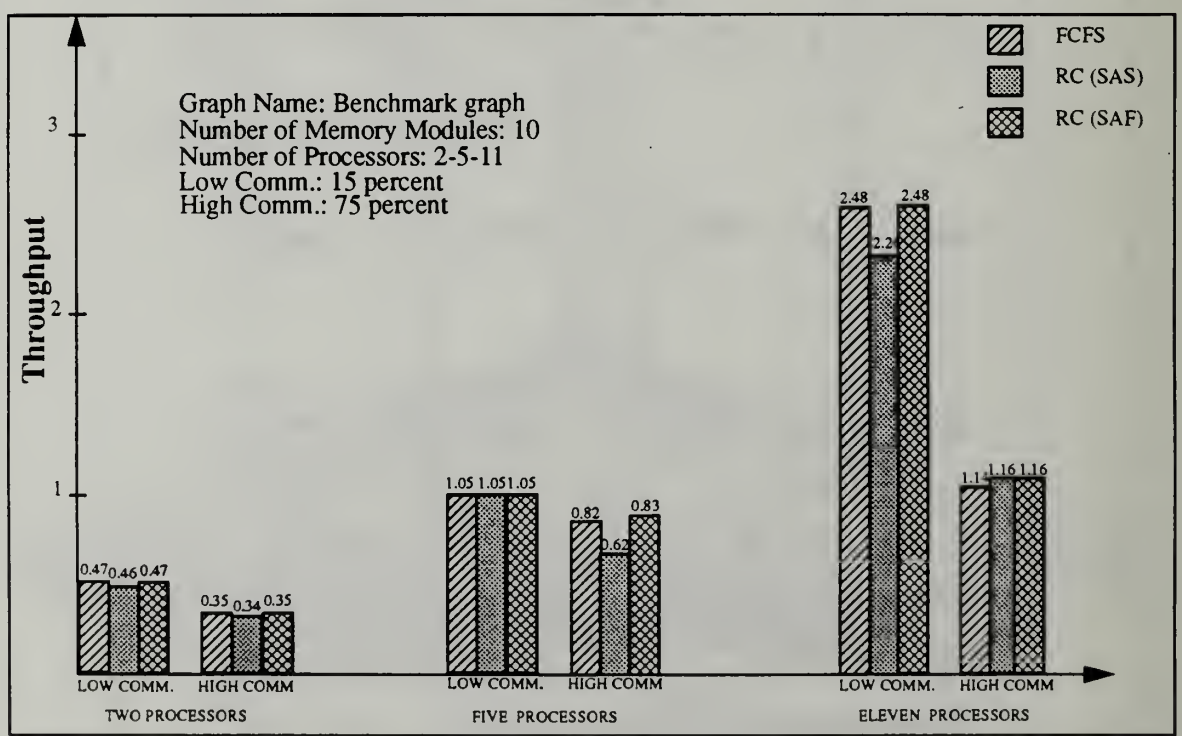


Figure 4.10. The throughput obtained from the benchmark graph.

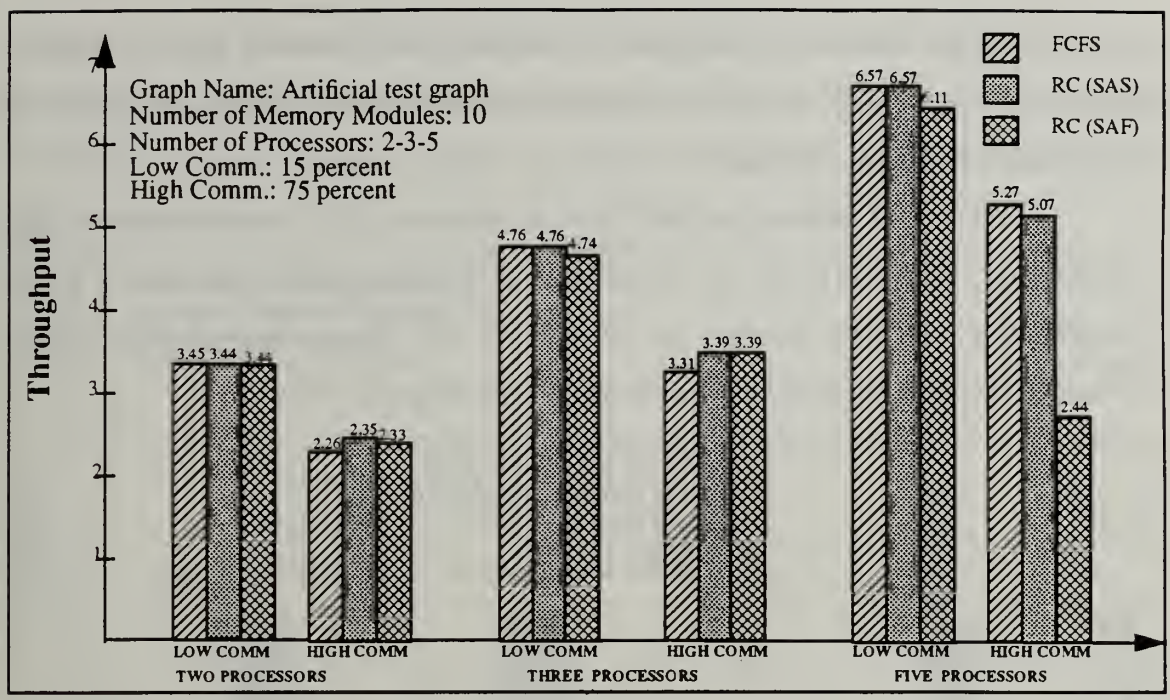


Figure 4.11 The throughput obtain from the artificial test graph.

In Figure 4.11, for high communication overheads, SAS demonstrates better throughput in most cases, but for low communication SAS and FCFS are exhibiting almost same throughput.

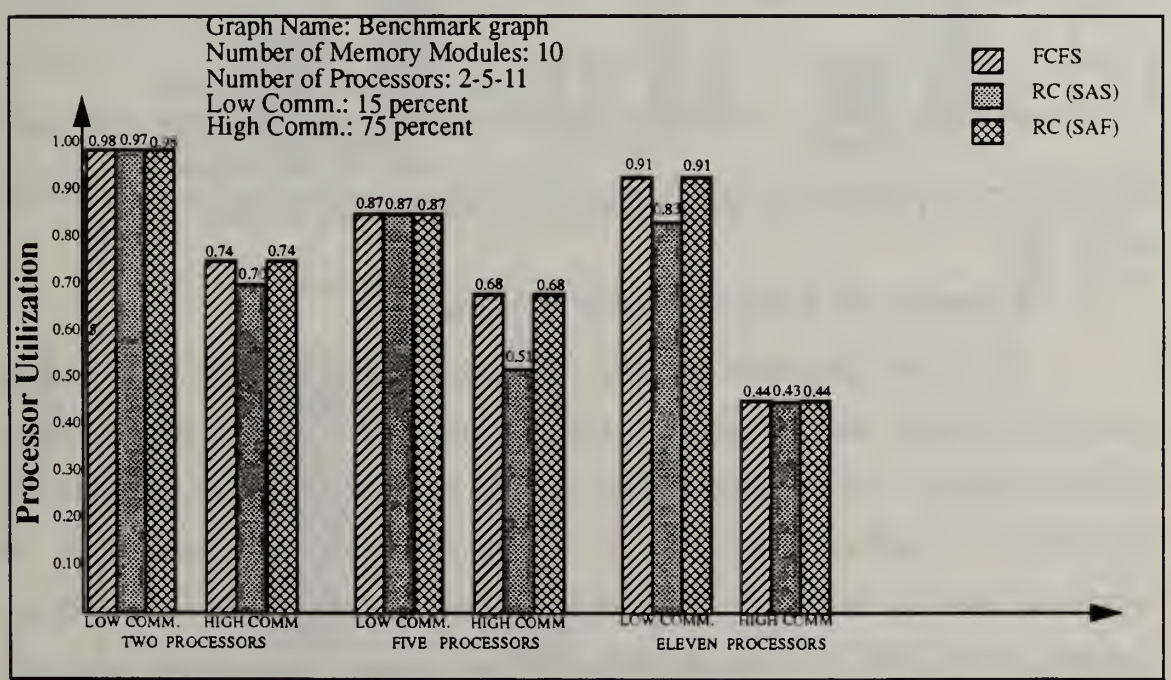


Figure 4.12. The processor utilization obtained from benchmark graph.

Figure 4.12 shows the processor utilization of benchmark graph. The results demonstrate The FCFS scheduling has slightly better utilization than SAS and SAF scheduling techniques in all cases.

Figure 4.13 depicts the coefficient of variation (COV) of the iteration length distribution. This figure measures the regularity of the output production which is at most importance in real-time systems. As revealed by this figure, none of the scheduling technique is conclusively better than the others with respect to COV.

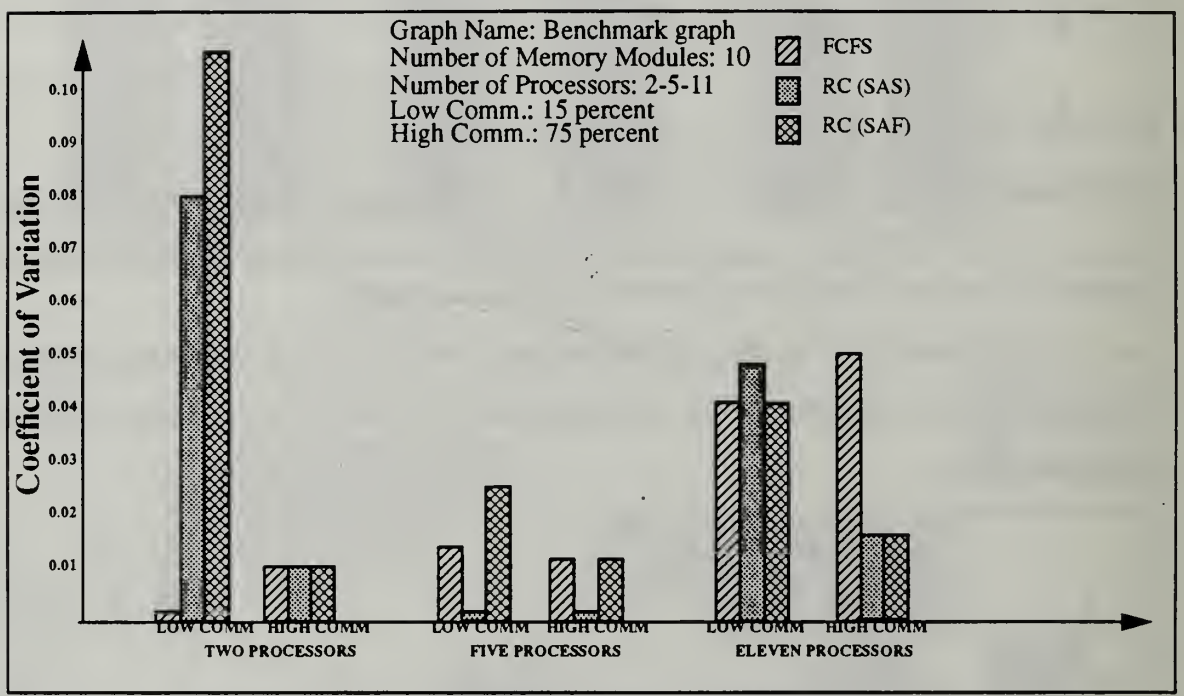


Figure 4.13. The Coefficient of Variation obtained from benchmark graph.

2. Evaluating the Effect of Memory Mapping

The second experiment is performed to see if mapping the queues to memory modules to minimize the access conflicts will give better results than the randomly mapped memory modules. The mapping is done maually by inspecting th RC schedule.The experiment is performed assuming 30 percent communication overheads. In the benchmark graph,10 memory modules and 5 processors are used, and the obtained results are depicted in Figure 4.14. In the artificial test graph,10 memory modules and 3 processors

are used, and the obtained results are shown in Figure 4.15. As revealed by the figures, mapping the queues to memories according to the cylinder assignment gives better throughput than the random mapped memories.(The relative change in throughput is minuscale, however).

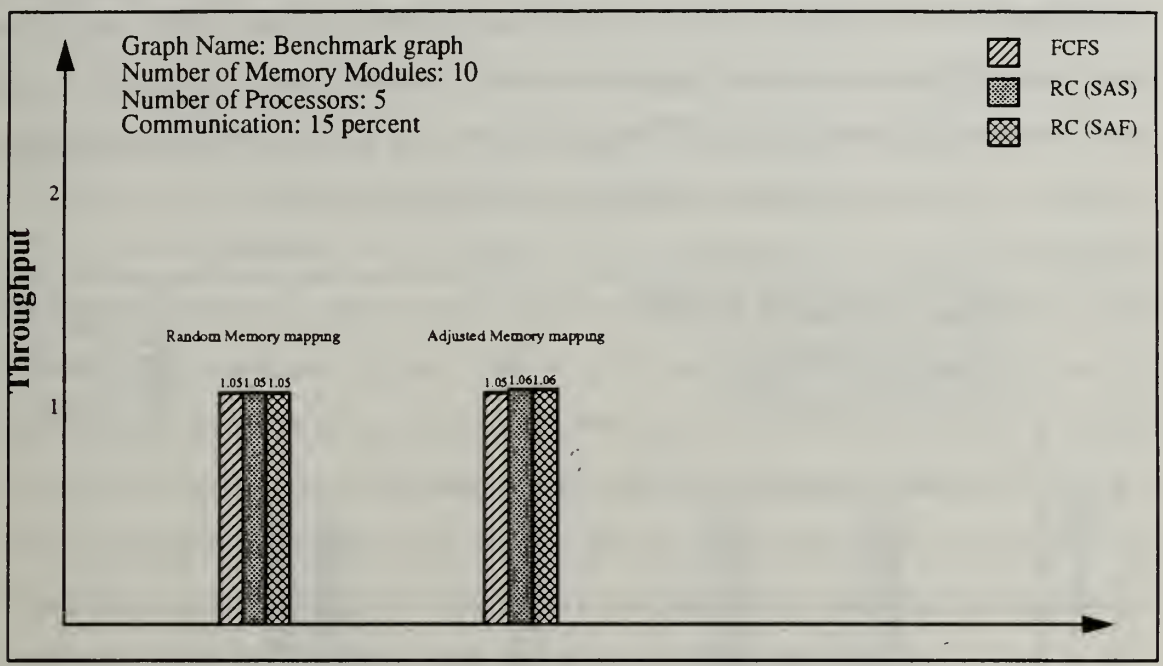


Figure 4.14. The effect of the memory mapping on scheduling techniques (Benchmark)

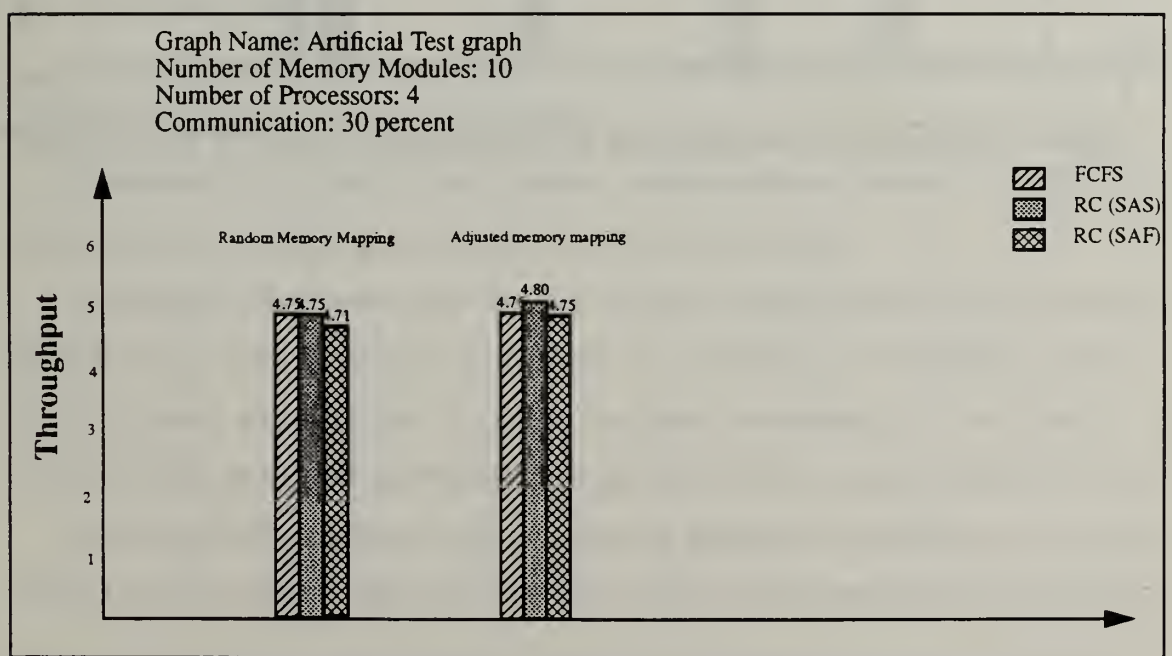


Figure 4.15 The effect of the memory mapping on scheduling techniques (Test graph)

3. Effect of the Queue Size on the System Throughput

This experiment is performed to watch the effect of changing the data queue sizes with respect to FCFS, and compare that the performance of SAS and SAF assuming the queue sizes dictated by the RC assignment. The experiment was performed on the artificial test graph with 4 processors and 10 memory modules. The data rate which saturates the machine in FCFS assuming very large queue sizes, is chosen. Different queue sizes were tried. The results are demonstrated in Figure 4.16. For the given data rate, the experiment results reveal that queue size does not effect the throughput for FCFS.

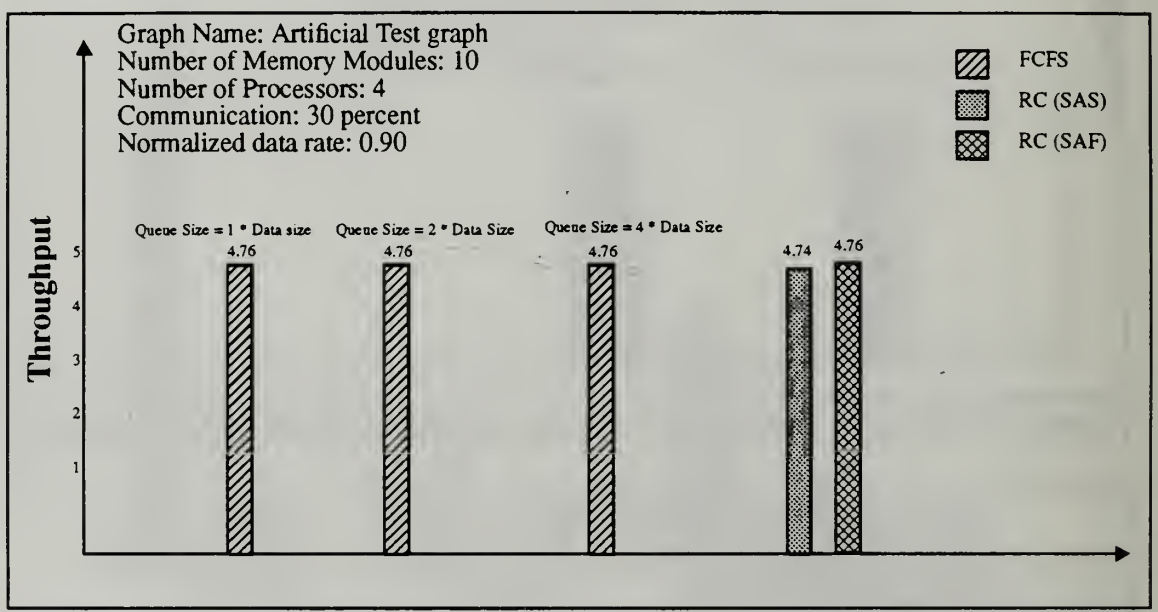


Figure 4.16 The effect of the data queue size on throughput with FCFS,SAS and SAF

V. CONCLUSION REMARKS

A. CONCLUSIONS

In this thesis, a simulator (PIPDAFS) which is used for Large Grain Data Flow machines, and a graph restructurer (GR) which determines the node dependencies according to a given strategy and restructures the original graph were developed. A series of experiments were performed by using GR and PIPDAFS. GR and PIPDAFS are a kind of tool that can be used to test the different scheduling strategies.

In our experiments, we compared FCFS scheduling with two other scheduling techniques in which the graphs were restructured and the nodes were executed according to their graph topologies. While in theory, restructuring techniques should improve the performance, the result of the experiments demonstrate that this improvement is meager. We believe that the reason for these results is that the mapping techniques employed in these experiments completely ignored the effects of communication. We believe that combining the restructuring techniques with a mapping, while considering communication overhead, would give results closer to the anticipated theoretical results.

B. FUTURE WORK

As noted from the experimental results, a new mapping algorithm which considers the communication overhead should be designed.

Algorithms for allowing the complete communication-computation overlap and deterministic nonconflicting resource usage should be identified.

Currently synchronization arcs that are obtained through restructuring are manually pruned. Some work is needed in the area of determining the minimal number of synchronization arcs required to satisfy the graph restructuring, as the number of synchronization arcs can be potentially be a major factor in the overall system performance.

In this research, a dynamic node to processor assignment technique was employed. Using a static processor assignment technique could be a step towards achieving more run

-time determinism. This should be critically weighed against the utility of fully dynamic processor assignment techniques with respect to fault tolerance.

APPENDIX A : Sample Graph File

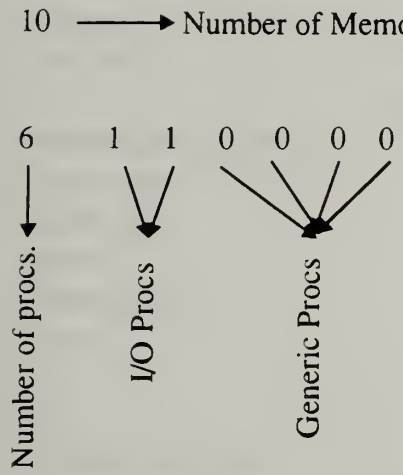
25 → Number of nodes

Node id	Node type	Aisize	Execution t.	Fix setup t.	Fix breakdown t.	Proc. Type	Alt. Proc. type
1	1	0	5000	-1	-1	1	1
7	1	0	3000	-1	-1	1	1
13	1	0	5000	-1	-1	1	1
19	1	0	3000	-1	-1	1	1
2	0	0	15000	-1	-1	0	0
8	0	1024	25000	-1	-1	0	0
14	0	1024	5000	-1	-1	0	0
20	0	1024	10000	-1	-1	0	0
3	0	1024	15000	-1	-1	0	0
9	0	1024	10000	-1	-1	0	0
15	0	1024	20000	-1	-1	0	0
21	0	1024	25000	-1	-1	0	0
4	0	1024	50000	-1	-1	0	0
10	0	1024	25000	-1	-1	0	0
16	0	1024	150000	-1	-1	0	0
22	0	1024	20000	-1	-1	0	0
5	0	1024	10000	-1	-1	0	0
11	0	1024	20000	-1	-1	0	0
17	0	1024	40000	-1	-1	0	0
23	0	1024	80000	-1	-1	0	0
6	0	1024	20000	-1	-1	0	0
12	0	1024	5000	-1	-1	0	0
18	0	1024	10000	-1	-1	0	0
24	0	1024	15000	-1	-1	0	0
25	2	256	3000	-1	-1	1	1

24 → Number of Queues

Queue id	Queue type	Node in	Node out	Threshold	Initial len.	Cons/prod. qty	Capacity
1	1	1	2	16384	0	16384	131072
2	1	2	9	16384	0	16384	131072
3	1	3	4	4096	0	4096	32768
4	1	4	5	4096	0	4096	32768
5	1	5	6	4096	0	4096	32768
6	1	6	25	4096	0	4096	32768
7	1	7	8	16384	0	16384	131072
8	1	8	9	16384	0	16384	131072
9	1	9	10	4096	0	4096	32768
10	1	10	11	4096	0	4096	32768
11	1	11	12	4096	0	4096	32768
12	1	12	25	4096	0	4096	32768
13	1	13	14	16384	0	16384	131072
14	1	14	15	16384	0	16384	131072
15	1	15	10	4096	0	4096	32768
16	1	16	17	4096	0	4096	32768
17	1	17	18	4096	0	4096	32768
18	1	18	25	4096	0	4096	32768
19	1	19	20	16384	0	16384	131072
20	1	20	15	16384	0	16384	131072
21	1	21	22	4096	0	4096	32768
22	1	22	23	4096	0	4096	32768
23	1	23	24	4096	0	4096	32768
24	1	24	25	4096	0	4096	32768
25	1	20	21	2048	0	2048	16384
26	1	8	3	4096	0	4096	32384
27	1	9	16	2048	0	2048	16384

Sample Machine Configuration File



APPENDIX B: Simulator Source Code

```
// Author : Cem Akin
// Advisor : Amr Zaky
// Description : LGDF machine simulator class header file
// Date : 12 November 1992
// Last Revised : 03 January 1993

#include"scheduler.h"
#include"mlist.h"
#include <fstream.h>

class simulator {
public :
        simulator();
        ~simulator();
        void simulate();
        void produce_readqueues(gnode*,event*);
        void produce_writequeues(gnode*,event*);
        void initialize_events(int);
        void process_event(event*,fstream&,fstream&,int);
        void consume_q(int);
        void printstatistics(fstream&,fstream&,int,int,double,double);
        void increasewriteamount(gnode *);
        void increasereadamount(gnode *);
        void decreasewriteamount(gnode *);
        void decreasereadamount(gnode *);
        boolean assign_processor(gnode *);
        boolean arealloutputqsready(gnode *);
        boolean areallqsot(gnode *);
        boolean arealloutputnodesexecuted();
        void initializeoutputnodes();

private :

        scheduler* sch;
        mlist * ml;
        plist* pl;
        Qlist * gqueuelist;
```

```
nlist * gnodelist;
pqueue * eventqueue;
int startinstance,
finishinstance,
inststart,
fixedcomm;

boolean onlyonecanbeready,
incremented;
double clock,
interval,
communication,
comm;
};

// Author : Cem Akin
// Advisor : Amr Zaky
// Description : LGDF machine simulator class.
// Date : 12 November 1992
// Last Revised : 03 January 1993
#include<iostream.h>
#include"simulator.h"
#include<math.h>

// Constructor of simulator class.
simulator::simulator(){
      clock = 0;
      sch = new scheduler;
      ml = new mlist;
      pl = new plist;
      startinstance = 0;
      finishinstance = 0;
      incremented = false;
      gqueuelist = new Qlist;
      gnodelist = new nlist;
      eventqueue = new pqueue;
      communication = 0.0;
      comm =1;
      fixedcomm =100;
};
// Destructor of simulator class
```

```
simulator::~simulator(){
      delete sch;
      delete ml;
      delete pl;
      delete gqueuelist;
      delete gnodelist;
      delete eventqueue;
      sch = NULL;
      ml = NULL;
       pl = NULL;
      gqueuelist = NULL;
      gnodelist = NULL;
      eventqueue = NULL;
};


// This function initialize the simulator to start the simulation.
void simulator::initialize_events(int period){
      nlist* l=gnodelist;
      event* tempevent;
      // For each input node produces an event that activates the simulator
      for(l->current=l->head;l->current;l->current=l->current->nextitem){
          if (l->current->element->ntype==input){
              l->current->element->dataperiod = period;
              tempevent = new event;
              tempevent->eventname=reach_production_period;
              tempevent->starttime=clock;
              tempevent->priority =clock;
              tempevent->nodenum =l->current->element->nodeid;
              eventqueue->enqueue(tempevent);
          };
      };
};


// This function reads the input files, gets user choices, initializes the
//simulator and
void simulator::simulate(){
     fstream myfile,
     startfile,
       endfile,
```

```
 userdata,
 inputperiod,
 config,
 mem,
 logfile;

 myfile.open("graph.dat",ios::in);
 config.open("machine.config",ios::in);
 mem.open("memodules",ios::in);
 int index,
 nummemos,
 choice,
 option,
 option2,
 instancenum,
 dperiod;
 double drate;
 intnode* tnode;
 event* tempevent;
startfile.open("starttimes",ios::out);
endfile.open("endtimes",ios::out);
tempevent = new event;
userdata.open("userdat",ios::in);
inputperiod.open("inperiod",ios::in);
userdata >> inststart;
userdata >> instancenum;
userdata >> option;
if (option == 1)
   onlyonecanbeready = true;
else
   onlyonecanbeready = false;
userdata >> choice;
userdata >> option2;
if (option2){
   logfile.open("log_file",ios::applios::out);
 };

 nummemos=ml->loadmemory(config);
 int pno;
 config >>pno;
  int pnum = pl->loadprocessors(config,pno);
```

```
    for (index=1;index<=pno;index++){
        tnode = new intnode;
        tnode->setnode(index,0);
        sch->freeproclist->enqueue(tnode);
     };

    int tottime = gnodelist->loadnodes(myfile,mem,choice,nummemos);
    gqueuelist->loadqueues(myfile,mem,gnodelist,nummemos);
    myfile.close();
    inputperiod >> dperiod;
    initialize_events(dperiod);
    drate = double (double(tottime / pnum)/dperiod);
    cerr << "\nSIMULATION IS IN PROCESS PLEASE WAIT !!!!\n";

    while ((! (eventqueue->empty()))&& (instancenum>finishinstance)) {
          tempevent = eventqueue->dequeue();
          process_event(tempevent,startfile,endfile,instancenum);
          if (option2){
             logfile << clock << " ";
             tempevent->printevent(logfile);
           };
    };

    startfile.close();
    endfile.close();
    inputperiod.close();
    userdata.close();
    config.close();
    logfile.close();

    printstatistics(startfile,endfile,instancenum-inststart+1,instancenum,drate,tottime);
};

// This function processes the given event and enqueue the produced events to
// event queue.
void simulator::process_event(event *e,fstream& startfile,
    fstream& endfile,int instancenum){
    event * tempevent=new event;
    gnode * tempnode ;
     QueueItem * tempqueue;
```

```
memory * tempmemory;
processor * tempproc;
intnode *tnode =new intnode;
int duration;
clock = e->starttime; //Advance the system clock to event starttime
switch (e->eventname){

//if input data period has been reached
case reach_production_period:{
    tempnode=gnodelist->getnode(e->nodenum);
    tempevent->eventname=inqsoverthreshold; //Produce event that
    tempevent->starttime=clock; //indicates input queues
    tempevent->priority =clock; //are overthreshold.
    tempevent->nodenum = e->nodenum; //It is assumed that amount
    eventqueue->enqueue(tempevent); //of data at each period is
    //large enough to make input
    //queues overthreshold

    tempevent = new event; //Produce next data period event
    tempevent->eventname=reach_production_period;
    tempevent->starttime=clock+tempnode->getdatarate();
    tempevent->priority = tempevent->starttime;
    tempevent->nodenum = e->nodenum;
    eventqueue->enqueue(tempevent);
    break;
};

//if all node input queues are overthreshold
case inqsoverthreshold:{
    tempnode = gnodelist->getnode(e->nodenum);
    if (arealloutputqsready(tempnode)||tempnode->getnodetype()==output){
        tempevent->eventname = ready_node;//If all node output queues
        tempevent->starttime = clock; //have space or node is an
        tempevent->priority = clock; //output node,then produce
        tempevent->nodenum = e->nodenum; //node is ready event
        eventqueue->enqueue(tempevent);
    }
    else {
        if (tempnode->ntype==instruction){
            tempnode->waitingforoutput = true;//Otherwise set the flag to
        tempnode->waitingnumber++; //indicate node is waiting for
```

```
    };
  };
  break;
};

//if node is ready to be executed
case ready_node:{
    tempnode = gnodelist->getnode(e->nodenum);
    //If node can not be put into the ready list decrease the readamount
    //which is previously incremented to prevent
    if (onlyonecanbeready && (sch->member(e->nodenum)))
       decreasereadamount(tempnode);
    //If node is not already in rl then put node to rl and schedule a
    //from rl.
   if (onlyonecanbeready && (!(sch->member(e->nodenum)))){
       sch->putnodeinrl(gnodelist->getnode(e->nodenum));
       increasewriteamount(tempnode);
   }
   else{ //If node is already in rl increment number of waiting ready
        //nodes
        if (onlyonecanbeready)
           tempnode->readycount++;
        else{ // If more than one node can be ready at any time
             //put node to rl. and schedule a node
             sch->putnodeinrl(gnodelist->getnode(e->nodenum));
             increasewriteamount(tempnode);
         };
   };
   tempevent = new event;
   tempevent->eventname = schedule_a_node_from_rl;
   tempevent->starttime = clock;
   tempevent->priority = clock;
   eventqueue->enqueue(tempevent);
    break;
 };
 //If rl is not empty schedule a node from rl.
 case schedule_a_node_from_rl:{
     if (!(sch->emptyrl())){
        if (!(sch->isbusy()))
            sch->schedule_node(gnodelist,pl,eventqueue,clock);
  };
```

```
break;
 };
//If node setup is started
case start_setup :{
     tempnode=gnodelist->getnode(e->nodenum);
     tempproc = pl->getproc(e->assocproc);
     if (!(incremented))
        if (tempnode->getnodetype()==input&&
           startinstance<=tempnode->getnodeinstance()){
            incremented = true;
            startinstance++;
            if (startinstance >= inststart)
               startfile << startinstance << " " << clock <<endl;
            if (startinstance == inststart)
               interval = clock;
        };

        if ((tempnode->getsetuptime()>=0)){
            if ((!(tempproc->isbreakdownbusy()))&&
               (!(tempproc->isexecbusy()))){
                  tempproc->startutil=clock;
             };

             tempproc=pl->getproc(e->assocproc);

             if ((!(tempproc->isbreakdownbusy()))||
                  tempnode->getsetuptime()==0){ // If breakdown is not busy
                 tempproc->setsetupbusytill(clock+tempnode->getsetuptime());
                 tempproc->setsetupbusy(true);
                 tempevent = new event;
                 tempproc->waitingonexec = false;
                 incremented = false;
                 tempevent->eventname = finish_setup;
                 tempevent->starttime = clock+tempnode->getsetuptime();
                 tempevent->priority = tempevent->starttime;
                 tempevent->nodenum = e->nodenum;
                 tempevent->queuenum = e->queuenum;
                 tempevent->assocproc = e->assocproc;
                 tempevent->assocmem = e->assocmem;
                 eventqueue->enqueue(tempevent);
          }
```

```
        else{ // Otherwise start to read after setup finished
            tempevent = new event;
            tempevent->eventname = start_setup;
            tempevent->starttime = tempproc->getbreakdownbusytime();
            tempevent->priority = tempevent->starttime;
            tempevent->nodenum = e->nodenum;
            tempevent->assocproc = e->assocproc;
            tempevent->assocmem = tempnode->memid;
             eventqueue->enqueue(tempevent);
         };
 }
else{
    if ((!(tempproc->isbreakdownbusy()))&&
      (!(tempproc->isexecbusy()))){
        tempproc->startutil=clock;
     };
     tempproc=pl->getproc(e->assocproc);
     if (!(tempproc->isbreakdownbusy())){ // If breakdown is not busy
         tempevent = new event; //start to read primitive inst
         tempproc->setsetupbusytill(clock+tempnode->getsetuptime());
         tempproc->setsetupbusy(true);
         tempproc->waitingonexec = false;
         incremented = false;
         tempevent->eventname = start_reading_instruction_stream;
         tempevent->starttime = clock;
         tempevent->priority = clock;
         tempevent->nodenum = e->nodenum;
         tempevent->assocproc = e->assocproc;
         tempevent->assocmem = tempnode->memid;
         eventqueue->enqueue(tempevent);
      }
       else{ // Otherwise start to read after setup finished
            tempevent = new event;
            tempevent->eventname = start_reading_instruction_stream;
            tempevent->starttime = tempproc->getbreakdownbusytime();
            tempevent->priority = tempevent->starttime;
            tempevent->nodenum = e->nodenum;
            tempevent->assocproc = e->assocproc;
            tempevent->assocmem = tempnode->memid;
            eventqueue->enqueue(tempevent);
   };
```

```
};
break;
};

//Start to read primitive instruction
case start_reading_instruction_stream:{
tempevent = new event;
tempmemory = ml->getmem(e->assocmem);
tempproc = pl->getproc(e->assocproc);
 tempnode = gnodelist->getnode(e->nodenum);
  if (!(tempmemory->isbusy())){ // if memory is not busy
 //Calculate reading delay and finish reading after this delay
     duration = (int (fixedcomm + comm*tempnode->aissize));
   tempproc->setsetupbusytill(clock+duration);
   tempmemory->setbusy(true);
   tempmemory->setbusytill(clock+duration);
   tempevent->eventname = finish_reading_instruction_stream;
   tempevent->starttime = clock+duration;
   tempevent->priority = clock+duration;
   tempevent->nodenum = e->nodenum;
   tempevent->assocproc = e->assocproc;
   tempevent->assocmem = e->assocmem;
    eventqueue->enqueue(tempevent);
 }
 else { //otherwise try to read after memory become available
     tempproc->setsetupbusytill(tempmemory->getbusytime());
     tempevent = new event;
     tempevent->eventname = start_reading_instruction_stream;
     tempevent->starttime = tempmemory->getbusytime();
     tempevent->priority = tempevent->starttime;
     tempevent->nodenum = e->nodenum;
     tempevent->assocproc = e->assocproc;
     tempevent->assocmem = e->assocmem;
     eventqueue->enqueue(tempevent);
 };
 break;
 };

 //finishes the reading of primitive instructions
 case finish_reading_instruction_stream :{
     tempmemory = ml->getmem(e->assocmem);
```

```
tempnode = gnodelist->getnode(e->nodenum);
tempmemory->setbusy(false);
//Produce read queues for output queues
produce_readqueues(tempnode,e);
break;
 };

// Starts to read specified input queue if processor is not already
//reading and memory to be read from is not busy
case start_readqueue:{
   tempqueue = gqueuelist->getqueue(e->queuenum);
   tempmemory= ml->getmem(tempqueue->getgmid());
   tempproc=pl->getproc(e->assocproc);
   if (tempqueue->qtype == syn_arc){
      tempevent = new event;
      tempevent->eventname=finish_reading;
      tempevent->starttime=clock;
      tempevent->priority=tempevent->starttime;
      tempevent->queuenum=e->queuenum;
      tempevent->nodenum=e->nodenum;
      tempevent->assocproc=e->assocproc;
      tempevent->assocmem = tempqueue->getgmid();
      eventqueue->enqueue(tempevent);
    }
   else{
      if (!(tempmemory->isbusy())){
        if (!(tempproc->readinginprocess)){
           tempproc->readinginprocess = true;
               duration =(int(fixedcomm+comm*tempqueue->consumptionqty));
           tempproc->setsetupbusytill(clock+duration);
           tempmemory->setbusy(true);
           tempmemory->setbusytill(clock+duration);
           tempevent->eventname=finish_reading;
           tempevent->starttime=clock+duration;
           tempevent->priority=tempevent->starttime;
           tempevent->queuenum=e->queuenum;
           tempevent->nodenum=e->nodenum;
           tempevent->assocproc=e->assocproc;
           tempevent->assocmem = tempmemory->getmemid();
           eventqueue->enqueue(tempevent);
   }
```

```
        else{ //if processor is already reading
            tempevent->eventname=start_readqueue;
            tempevent->starttime=tempproc->setupbusytill;
            tempevent->priority=tempevent->starttime;
            tempevent->nodenum=e->nodenum;
            tempevent->queuenum=e->queuenum;
            tempevent->assocproc=e->assocproc;
            eventqueue->enqueue(tempevent);
          };
        }
        else{ //If memory to be read is busy
            tempproc->setsetupbusytill(tempmemory->getbusytime());
            tempevent->eventname = e->eventname;
            tempevent->starttime = tempmemory->getbusytime();
            tempevent->priority = tempevent->starttime;
            tempevent->queuenum = e->queuenum;
            tempevent->nodenum = e->nodenum;
            tempevent->assocproc = e->assocproc;
            tempevent->assocmem = tempmemory->getmemid();
            eventqueue->enqueue(tempevent);
        };
      };
      break;
      };

      //Mark processor as not busy and look if there are waiting ready
      //nodes for a processor.
      case free_processor:{
          tempproc = pl->getproc(e->assocproc);
          tempproc->setfree(true);
          tnode->setnode(tempproc->procid,0);
          sch->freeproclist->enqueue(tnode);
          tempevent->eventname = schedule_a_node_from_rl;
          tempevent->starttime = clock;
          tempevent->priority = clock;
          eventqueue->enqueue(tempevent);
      break;
      };


//Starts execution
```

```
case start_execution : {
    tempnode=gnodelist->getnode(e->nodenum);
    //Get the execution delay
    duration=tempnode->getexectime();
    tempproc=pl->getproc(e->assocproc);
    tempevent = new event;
    tempevent->eventname = start_execution;
    tempevent->starttime = tempproc->getexecbusytime();
    tempevent->priority = tempevent->starttime;
    tempevent->nodenum = e->nodenum;
    tempevent->assocproc = e->assocproc;
    eventqueue->enqueue(tempevent);
    break;
 };

 //Finishes execution if breakdown and setup stages are not busy
 case finish_execution :{
     tempnode=gnodelist->getnode(e->nodenum);
     tempproc=pl->getproc(e->assocproc);
     if(!(tempproc->isbreakdownbusy())){
          if (!(tempproc->issetupbusy())){
              tempproc->setexecbusy(false);
              tempproc->setbreakdownbusy(true);
              tempproc->setbreakdownbusytill(clock);
              tempnode->lastinstance++;
              tempevent->eventname = start_breakdown;
               tempevent->starttime = clock;
               tempevent->priority = clock;
               tempevent->nodenum = e->nodenum;
               tempevent->assocproc = e->assocproc;
               eventqueue->enqueue(tempevent);
          }
          else{//Otherwise indicate that execution is waiting for setup
               tempproc->waitingonsetup = true;

          if (tempproc->waitingonexec) { //This part prevents the deadlock
              tempproc->setsetupbusy(false);
              tempproc->setexecbusy(false);
              tempproc->waitingonsetup=false;
              tempevent->starttime=clock;
           }
```

```
            else
              tempevent->starttime= tempproc->getsetupbusytime();

            tempproc->setexecbusytill(tempevent->starttime);
            tempevent->eventname = finish_execution;
            tempevent->priority = tempevent->starttime;
            tempevent->nodenum = e->nodenum;
            tempevent->assocproc = e->assocproc;
            eventqueue->enqueue(tempevent);
        };
    }
    else{ //If breakdown is busy, try to finish execution
        tempproc->setexecbusytill(tempproc->getbreakdownbusytime());
        tempevent->eventname = finish_execution;
        tempevent->starttime = tempproc->getbreakdownbusytime();
        tempevent->priority = tempevent->starttime;
        tempevent->nodenum = e->nodenum;
        tempevent->assocproc = e->assocproc;
        eventqueue->enqueue(tempevent);
    };
    break;
  };

  //Starts to breakdown and produces start write queue
  case start_breakdown :{
      tempnode=gnodelist->getnode(e->nodenum);
      tempproc=pl->getproc(e->assocproc);
      tempproc->setbreakdownbusytill(clock+tempnode->getbreakdowntime());
      tempproc->setbreakdownbusy(true);
      //Produces write queues
      produce_writequeues(tempnode,e);
      break;
  };

//Starts writing to the specified GM if processor is not already
//writing and memory to be written is not busy
case start_write_queue:{
    tempqueue=gqueuelist->getqueue(e->queuenum);
    tempproc=pl->getproc(e->assocproc);
    if (tempqueue->qtype==syn_arc){
    tempevent = new event;
```

```
    tempevent->eventname = finish_writing;
    tempevent->starttime = clock;
    tempevent->priority = tempevent->starttime;
    tempevent->nodenum = e->nodenum;
    tempevent->queuenum = e->queuenum;
    tempevent->assocproc = e->assocproc;
    tempevent->assocmem = tempqueue->getgmid();
    eventqueue->enqueue(tempevent);
  }
 else{
     tempnode = gnodelist->getnode(e->nodenum);
      if (tempnode->getbreakdowntime()>=0){
          tempproc->setbreakdownbusytill(clock+tempnode->getbreakdowntime());
        tempmemory=ml->getmem(tempqueue->getgmid());
        tempevent->eventname = finish_writing;
        tempevent->starttime = clock+tempnode->getbreakdowntime();
        tempevent->priority = tempevent->starttime;
        tempevent->nodenum = e->nodenum;
        tempevent->queuenum = e->queuenum;
        tempevent->assocproc = e->assocproc;
        tempevent->assocmem = tempmemory->getmemid();
        eventqueue->enqueue(tempevent);
       }
      else{
        //Calculate the write process delay
        duration =(int(fixedcomm+comm*tempqueue->getproductionqty()));
        tempmemory=ml->getmem(tempqueue->getgmid());
        if ((!(tempmemory->isbusy()))||tempnode->getnodetype()==input){
            if (!(tempproc->writeinprocess)){
                tempproc->writeinprocess = true;
                tempproc->setbreakdownbusytill(clock+duration);
                 if (tempnode->getnodetype()==inst){
                    tempmemory->setbusy(true);
                    tempmemory->setbusytill(duration+clock);
                  };
                  tempevent->eventname = finish_writing;
                  tempevent->starttime = clock+duration;
                  tempevent->priority = tempevent->starttime;
                  tempevent->nodenum = e->nodenum;
                  tempevent->queuenum = e->queuenum;
                     tempevent->assocproc = e->assocproc;
```

```
                tempevent->assocmem = tempqueue->getgmid();
                eventqueue->enqueue(tempevent);
              }
              else{
                tempevent->eventname = start_write_queue;
                tempevent->starttime = tempproc->breakdownbusytill;
                tempevent->priority = tempevent->starttime;
                tempevent->nodenum = e->nodenum;
                tempevent->queuenum = e->queuenum;
                tempevent->assocproc = e->assocproc;
                eventqueue->enqueue(tempevent);
              };
            }
            else{
              tempproc->setbreakdownbusytill(tempmemory->getbusytime());
             tempevent->eventname = e->eventname;
             tempevent->starttime = tempmemory->getbusytime();
             tempevent->priority = tempevent->starttime;
             tempnode = gnodelist->getnode(e->nodenum);
             tempevent->nodenum = e->nodenum;
             tempevent->queuenum = e->queuenum;
             tempevent->assocproc = e->assocproc;
             tempevent->assocmem = tempqueue->getgmid();
             eventqueue->enqueue(tempevent);
            };
        };
    };
    break;
};

//Finishes reading
case finish_reading:{
      tempproc=pl->getproc(e->assocproc);
      tempproc->readinginprocess=false;
      tempnode=gnodelist->getnode(e->nodenum);
      tempmemory=ml->getmem(e->assocmem);
      tempmemory->setbusy(false);
      tempnode->incnumreadqs();
      //If this is the last queue to be read, finish setup
     if (tempnode->lastnoderead()){
    tempnode->setnumreadqs();
```

```
        consume_q(e->queuenum);
        tempevent->eventname = finish_setup;
        tempevent->starttime = clock;
        tempevent->priority = clock;
        tempevent->nodenum = e->nodenum;
        tempevent->queuenum = e->queuenum;
        tempevent->assocproc = e->assocproc;
        tempevent->assocmem = e->assocmem;
        eventqueue->enqueue(tempevent);
    };
    break;
};

//Finishes the node setup if execution stage is not busy
case finish_setup:{
    tempproc=pl->getproc(e->assocproc);
    tempnode=gnodelist->getnode(e->nodenum);
    if(!(tempproc->isexecbusy())){
        //Free the processor
        tempproc->setsetupbusy(false);
        tempevent->eventname=free_processor;
        tempevent->starttime=clock;
        tempevent->priority =clock;
        tempevent->assocproc=e->assocproc;
        eventqueue->enqueue(tempevent);
        tempproc->onlyexectime = tempproc->onlyexectime +
                                 tempnode->getexectime();
        tempevent = new event;
        tempevent->eventname=start_execution;
        tempevent->starttime = clock;
        tempevent->priority = clock;
        tempevent->nodenum = e->nodenum;
        tempevent->assocproc = e->assocproc;
        eventqueue->enqueue(tempevent);
    }
    else{// Execution stage is busy try to finish setup
        //When execution is finished
        tempproc->waitingonexec = true;

    if (tempproc->waitingonsetup) { //This part prevents deadlock
        tempproc->setexecbusy(false);
```

```
        tempproc->waitingonsetup=false;
        tempproc->waitingonexec=false;
        tempevent->starttime=clock;
      }
      else
        tempevent->starttime=tempproc->getexecbusytime();

      tempproc->setsetupbusytill(tempevent->starttime);
      tempevent->eventname = finish_setup;
      tempevent->priority = tempevent->starttime;
      tempevent->nodenum = e->nodenum;
      tempevent->assocproc = e->assocproc;
      eventqueue->enqueue(tempevent);
    };
    break;
};

//Finishes writing process
case finish_writing:{
    tempproc=pl->getproc(e->assocproc);
    tempproc->writeinprocess=false;
    tempqueue=gqueuelist->getqueue(e->queuenum);
    tempnode =gnodelist->getnode(e->nodenum);
    tempnode->incnumwriteqs();
    tempmemory=ml->getmem(e->assocmem);
    tempmemory->setbusy(false);
    //Increment current length of queue
     tempqueue->inccurrentlength(tempqueue->getproductionqty());
     //Decrement writeamount since queue is physically written
    tempqueue->writeamount=tempqueue->writeamount-
    tempqueue->getproductionqty();

   //If this is the last queue to be written finish breakdown
   if (tempnode->lastqueuewrite()){
      tempnode->setnumwriteqs();
      tempevent->eventname=finish_breakdown;
      tempevent->starttime = clock;
      tempevent->priority = clock;
      tempevent->nodenum = e->nodenum;
     tempevent->queuenum = e->queuenum;
     tempevent->assocproc = e->assocproc;
```

```
    tempevent->assocmem = e->assocmem;
    eventqueue->enqueue(tempevent);
   };

  //If produeced data makes the queue overthreshold
   if (tempqueue->isoverthreshold()){
       tempevent = new event;
       tempevent->eventname=queue_overthreshold;
       tempevent->starttime=clock;
       tempevent->priority=clock;
       tempevent->nodenum=tempqueue->getsinknode();
       tempevent->queuenum=e->queuenum;
       eventqueue->enqueue(tempevent);
    };
    break;
};

//Finishes the node breakdown
case finish_breakdown:{
      tempnode = gnodelist->getnode(e->nodenum);
      tempproc=pl->getproc(e->assocproc);
      tempproc->setbreakdownbusy(false);
      if ((((!(tempproc->issetupbusy()))&&
          (!(tempproc->isexecbusy())))||
            finishinstance==(instancenum-1)){
            tempproc->durationprocbusy=tempproc->durationprocbusy+
            clock-tempproc->startutil;
            tempproc->startutil=clock;
      };
      tempevent = new event;
      tempnode=gnodelist->getnode(e->nodenum);
      tempnode->processing = false;
      tempevent = new event;
       if (tempnode->getnodetype()==output)
          tempnode->executed = true;
       if(arealloutputnodesexecuted()){
          initializeoutputnodes();
       finishinstance++;
       if (finishinstance >= inststart)
         endfile << finishinstance << " " << clock << endl;
       };
```

```
        //If ready node list is not empty
        if (!(sch->emptyrl())){
            tempevent->eventname = schedule_a_node_from_rl;
            tempevent->starttime = clock;
            tempevent->priority = clock;
            eventqueue->enqueue(tempevent);
        };
        break;
    };

    // If queue is overthreshold check the other input queues
    //If all overthreshold indicate node's input data is ready
    case queue_overthreshold:{
        tempnode=gnodelist->getnode(e->nodenum);
        if (areallqsot(tempnode)){
            increasereadamount(tempnode);
            tempevent->eventname=inqsoverthreshold;
            tempevent->starttime=clock;
            tempevent->priority =clock;
            tempevent->nodenum = e->nodenum;
            eventqueue->enqueue(tempevent);
        };
        break;
    };

    //Marks scheduler as not busy
    case free_scheduler:{
        sch->setbusy(false);
        //Checks if there is a ready node which is waiting for
        //scheduler try to schedule a node from rl
        tempevent->eventname = schedule_a_node_from_rl;
        tempevent->starttime = clock;
        tempevent->priority = tempevent->starttime;
        eventqueue->enqueue(tempevent);
        break;
    };
    case none:{
    break;
};
};
```

```
}; //end of procesevent


// Produces read queue events for each input if it is not an input node.
void simulator::produce_readqueues(gnode *n,event* e){
    intlist *l;
    event * tempevent ;
    l = n->getinputqueuelist();
    //If it is an input queue, then after fixed amount delay finish setup
    if (n->getnodetype() == input) {
      tempevent=new event;
      tempevent->eventname = finish_setup;
      tempevent->starttime = clock;
      tempevent->priority = tempevent->starttime;
      tempevent->nodenum = e->nodenum;
      tempevent->assocproc = e->assocproc;
      eventqueue->enqueue(tempevent);
    }
   else{
     //For internal nodes produce read queues
     for(l->current=l->head;l->current;l->current=l->current->nextnode){
        tempevent=new event;
        tempevent->eventname = start_readqueue;
        tempevent->starttime = clock;
       tempevent->priority = clock;
       tempevent->nodenum = n->getnodeid();
       tempevent->queuenum = l->current->number;
       tempevent->assocproc = e->assocproc;
       eventqueue->enqueue(tempevent);
      };
   };
};

//Produces write queue events for each output queue if node is not an
//output node
void simulator::produce_writequeues(gnode *n,event* e){
     intlist *l;
     event * tempevent;
     l = n->getoutputqueuelist();
     //If node is an output node finish breakdown after fixed amount of delay
     if (n->getnodetype() == output) {
```

```
        tempevent=new event;
        tempevent->eventname = finish_breakdown;
        tempevent->starttime = clock;
        tempevent->priority = tempevent->starttime;
        tempevent->nodenum = e->nodenum;
        tempevent->assocproc = e->assocproc;
       eventqueue->enqueue(tempevent);
     }
    else{
      //If node is an internal node produce write queue event for each output
      //queue
     for(l->current=l->head;l->current;l->current=l->current->nextnode){
        tempevent = new event;
        tempevent->eventname = start_write_queue;
        tempevent->starttime = clock;
        tempevent->priority = clock;
        tempevent->nodenum = n->getnodeid();
        tempevent->queuenum = l->current->number;
        tempevent->assocproc = e->assocproc;
        eventqueue->enqueue(tempevent);
      };
   };
};

//Consumes the input queues by decreasing the threshold amount
void simulator::consume_q(int num){
    intlist *l;
    QueueItem *temp;
     boolean flag;
    gnode *n,*m;
    temp=gqueuelist->getqueue(l->current->number);
    n=gnodelist->getnode(temp->nodeout);
    flag = temp->isthereenoughspace();
    temp->currentlength=temp->currentlength-temp->getthreshold();
    temp->readamount=temp->readamount-temp->getthreshold();
    //If consumption makes room for the waiting nodes
     if (!flag){
            //get source node
            m = gnodelist->getnode(temp->nodein);
            if (temp->isthereenoughspace()){
              //If all output qs are ready and if source node is waiting
```

```
                //for the output become ready
                if (arealloutputqsready(m)&& m->waitingforoutput){
                    m->waitingnumber--;
                    if (m->waitingnumber == 0)
                        m->waitingforoutput = false;
                    tempevent = new event;
                    tempevent->eventname=ready_node;
                    tempevent->starttime=clock;
                    tempevent->priority=clock;
                    tempevent->nodenum=m->getnodeid();
                    eventqueue->enqueue(tempevent);
                };
            };
        };
        //If sink node still overthreshold and it is not an input node
        //indicates that input queues are still overthreshold
        if (areallqsot(n)&&(n->getnodetype()!=input)){
          increasereadamount(n);
          tempevent = new event;
          tempevent->eventname=inqsoverthreshold;
          tempevent->starttime=clock;
          tempevent->priority=clock;
          tempevent->nodenum=n->getnodeid();
          eventqueue->enqueue(tempevent);
        };
};

//Returns boolean if all input queues are overthreshold
boolean simulator::areallqsot(gnode *n){
    QueueItem* q;
    intlist* ql = n->getinputqueuelist();
    for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
        q = gqueuelist->getqueue(ql->current->number);
        if ((q->currentlength-q->readamount)< q->thresholdqty)
            return false;
      };
      return true;
 };

//Increases the read amount of input nodes. It is used to prevent, input queues
//becomeoverthreshold even it is put in to ready list
```

```
void simulator::increasereadamount(gnode *n){
    QueueItem* q;
     intlist* ql = n->getinputqueuelist();
    for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
        q = gqueuelist->getqueue(ql->current->number);
        q->readamount = q->readamount +q->consumptionqty;
     };
};

//Decreases the read amount of input queues
void simulator::decreasereadamount(gnode *n){
    QueueItem* q;
    intlist* ql = n->getinputqueuelist();
    for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
        q = gqueuelist->getqueue(ql->current->number);
        q->readamount = q->readamount -q->consumptionqty;
     };
};

//Increases the write amount of output queues to prevent putting an extra ready
//node which will cause overcapacity
void simulator::increasewriteamount(gnode *n){
    QueueItem* q;
    intlist* ql = n->getoutputqueuelist();
    for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
        q = gqueuelist->getqueue(ql->current->number);
        q->writeamount = q->writeamount +q->productionqty;
     };
};

//Returns true if all output queues has enough space to store the result of
//source node execution
boolean simulator::arealloutputqsready(gnode *n){
     QueueItem* q;
     intlist* ql = n->getoutputqueuelist();
      for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
          q = gqueuelist->getqueue(ql->current->number);
          if ((q->capacity - q->currentlength-q->writeamount)< q->productionqty)
              return false;
       };
       return true;
```

```
};

//Return true if all output nodes of an instance is executed or not
boolean simulator::arealloutputnodesexecuted(){
        for(gnodelist->current= gnodelist->head;
            gnodelist->current;
            gnodelist->current=gnodelist->current->nextitem)
            if (gnodelist->current->element->getnodetype()==output)
                if (gnodelist->current->element->executed==false)
                    return false;
           return true;
};

//Initialize the output queues execution flag
void simulator::initializeoutputnodes(){
     for(gnodelist->current= gnodelist->head;
         gnodelist->current;
         gnodelist->current=gnodelist->current->nextitem)
         if (gnodelist->current->element->getnodetype()==output){
            gnodelist->current->element->exectimes--;
            if (gnodelist->current->element->exectimes==0)
                gnodelist->current->element->executed=false;
        };
};



//Prints the statistical information
void simulator::printstatistics(fstream& startfile,fstream& endfile,
                                int instnum,int endinst,double drate, double tottime){
          double end,
                 start,
                 difference,
                 sum,
                 squaresum,
               average,
               variance;
        double averageexectime =0.0;
        double averagebusytime =0.0;
        fstream aputil,thrput,lenvar,instlen,onlyexec;
         int cnt=0;
```

```
  cout << "*************************************************************\n";
  cout << "*************************************************************\n";
  cout << "** **\n";
  cout << "** STATISTICS **\n";
  cout << "** **\n";
  cout << "** **\n";
  cout << "*************************************************************\n";
  cout << "*************************************************************\n";
  cout << endl;
cout<<"PROCESSOR ID : PROCESSOR TYPE : PROCESSOR UTILIZATION(with comm:\n";
 for (pl->current=pl->head;pl->current;pl->current=pl->current->nextproc){
     cout <<" " << pl->current->p->procid << " ";
     cout << pl->current->p->ptype << " ";
     cout << double((pl->current->p->durationprocbusy))/double(clock) << endl;
     if(pl->current->p->gettype()!=io){
       averageexectime = averageexectime +
       double((pl->current->p->onlyexectime))/double(clock);
       averagebusytime = averagebusytime +
       double((pl->current->p->durationprocbusy))/double(clock);
       cnt++;
      };
 };

 for (gnodelist->current=gnodelist->head;
      gnodelist->current;gnodelist->current=gnodelist->current->nextitem)
     communication=communication+gnodelist->current->element->aissize*comm;

 for (gqueuelist->current=gqueuelist->head;
      gqueuelist->current;gqueuelist->current=gqueuelist->current->nextitem)
      communication=communication+
      gqueuelist->current->element->consumptionqty*comm;
     startfile.open("starttimes",ios::in);
    endfile.open("endtimes",ios::in);
    aputil.open("grphutil",ios::applios::out);
     onlyexec.open("grphexectime",ios::applios::out);
     instlen.open("grphresptime",ios::applios::out);
     lenvar.open("grphinstlenvar",ios::applios::out);
     thrput.open("grphthroughput",ios::applios::out);
     sum = 0;
     squaresum=0;
```

```
        double stime;
        for ( int loop=1;loop <=instnum;loop++){
              startfile >> start;
              startfile >> start;
              endfile >> end;
              endfile >> end;
              if (loop == 1)
                   stime = end;
              difference = end - start;
              sum = sum + difference;
               squaresum = squaresum + (difference * difference);
        };
        stime=end - stime;
        average = sum / double(instnum);
        cout << "DATA RATE :" << drate << endl;
        cout << "AVERAGE RESPONCE TIME :" << average << endl;
         variance = (squaresum-(double(instnum)*sqr(average)))/double(instnum);
         startfile.close();
         endfile.close();
         cout << "AVERAGE THROUGHPUT :" << ((instnum-1)*1000000)/stime<<endl;
         cout << "SIMULATION TIME :"<< clock << endl;
          cout << "INSTANCE LENGTH STANDARD VARIATION :" << sqrt(variance) << endl;
         cout << "COMMUNICATION OF ONE INSTANCE :"<<communication;
         cout << endl;
         cout << "COMPUTATION OF ONE INSTANCE :";
         cout << tottime<< endl;
         cout << "COMMUNICATION / COMPUTATION RATIO :";
         cout << communication / tottime;
         cout << endl;
         cout << endl;
         aputil << drate << " " << averagebusytime/double (cnt) <<endl;
        onlyexec<< drate << " " << averageexectime/double (cnt) << endl;
        instlen << drate << " " << average << endl;
        thrput << drate << " " << ((instnum-1)*1000000)/stime<<endl;
        lenvar << drate << " " << sqrt(variance)/average << endl;
      aputil.close();
     onlyexec.close();
     instlen.close();
     lenvar.close();
     thrput.close();
};
```

```
main(){
    simulator *s;
    s = new simulator;
    s->simulate();
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : Event Class header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>
#include <fstream.h>
#include "global.h"
#ifndef EVENT_H
#define EVENT_H

class event{
 public:
      event();
      event_type geteventname();
int getpriority();
      void printevent(fstream&);


      event_type eventname;
      double    starttime,
                priority;

      int       nodenum,
                queuenum,
                assocproc,
                assocmem;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : Event Class source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "event.h"

//Event class constructor
event::event(){
      eventname = none;
      starttime = 0.0;
      priority  = 0.0;
      nodenum   = 0;
      queuenum  = 0;
      assocproc = 0;
      assocmem  = 0;

};

//Returns the events priority
int event::getpriority(){
return priority;
};

//Returns event name
event_type event::geteventname(){
      return eventname;
};

//Prints the event to the log file
void event::printevent(fstream & logfile){
    logfile << eventname <<"   " << starttime << " " << priority << " " ;
    logfile << nodenum   << "   " << queuenum  << "    " << assocproc << " \n" ;
    logfile << assocmem <<endl;

};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : GNODE Class (Graph node).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <fstream.h>
#include "ilist.h"
#include "qlist.h"

#ifndef GNODE_H
#define GNODE_H

class gnode {
public:
     gnode();
     ~gnode();
     int loadnode(fstream&,int,int);
     int getnodeid();
     int getnodepriority();
     int getdataperiod();
     int getsetuptime();
     int getbreakdowntime();
     int getexectime();
     int getdatarate();
     void incnumreadqs();
     void incnumwriteqs();
     void setnumreadqs(){
        numreadqs=0;
  };
     void setnumwriteqs(){
        numwriteqs=0;
  };
     void incnodeinstance(){
        nodeinstance++;
  };
     int getnodeinstance(){
        return nodeinstance;
  };
     boolean lastnoderead();
     boolean lastqueuewrite();
```

```
        proctype getproctype();
        proctype getaltproctype();
        intlist* getinputqueuelist();
        intlist* getoutputqueuelist();
        friend ostream& operator<<(ostream&,gnode&);
        boolean operator<(gnode&);
        nodetype getnodetype();

//private:
        int nodeid,
            priority,
            memid,
            aissize,
            lastinstance,
            dataperiod,
            numreadqs,
            numwriteqs,
            totalinqs,
            totaloutqs,
            exectime,
            setuptime,
            breakdown,
            nodeinstance,
            waitingnumber,
            readycount,
            exectimes,
            order;

        double  proctime;
        boolean processing,
                waitingforoutput,
                executed;
               nodetype  ntype;
        proctype protype,
                 altproctype;
        intlist *inqlist,
                *outqlist;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : GNODE Class source code.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <stdlib.h>
#include "gnode.h"
#include <iostream.h>

//Gnode class contructor
gnode::gnode(){
      proctime = 0.0;
      nodeid = 0;
      priority=0;
      memid = 0;
      aissize=0;
      lastinstance=0;
      nodeinstance=-1;
      exectime=0;
      setuptime=0;
      breakdown=0;
      dataperiod=0;
      ntype = instruction;
      protype = inst;
      altproctype = inst;
      readycount = 0;
      numreadqs = 0;
      numwriteqs = 0;
      totalinqs = 0;
      totaloutqs = 0;
      waitingnumber = 0;
      inqlist =NULL;
      outqlist = NULL;
      processing = false;
      waitingforoutput = false;
      executed = false;
};

//Gnode class destructor
gnode::~gnode() {
```

```
    delete inqlist;
    delete outqlist;
    inqlist = NULL;
    outqlist = NULL;
};

//Loads a single graph node from given stream
int gnode::loadnode(fstream& grphfile,int pchoice, int gid){
int ttype,
      extime;

    srand(int(time(NULL)));
    grphfile >> nodeid;
    grphfile >> ttype;
    ntype = nodetype(ttype);
    memid = gid;
    grphfile >> aissize;
    grphfile >> exectime;
    extime = exectime;
    if (ttype != 0)
      extime = 0;
    grphfile >> setuptime;
    grphfile >> breakdown;
    grphfile >> ttype;


    switch (pchoice){
      case 1 :{
        priority = exectime;
        break;
      };
      case 2 :{
        priority = - aissize;
        break;
      };
      case 3 :{
        cout << "ENTER NODE PRIORITY :";
        cin >>priority;
        cout << endl;
        break;
      };
```

```
        case 4 :{
            priority = rand();
            break;
        };
        case 5 :
            break;
    };
    switch (ttype){
        case 0:{
            protype = inst;
            break;
        };
        case 1:{
            protype = io;
            break;
        };
        case 2:{
            protype = sec;
            break;
        };
        case 3:{
            protype = three;
            break;
        };
        case 4:{
            protype = four;
            break;
        };
        case 5:{
            protype = five;
            break;
        };
        case 6:{
            protype = six;
            break;
        };
        case 7:{
            protype = seven;
            break;
        };
        case 8:{
```

```
            protype = eight;
            break;
        };
    };
    grphfile >> ttype;
    switch (ttype){
        case 0:{
            altproctype = inst;
            break;
        };
        case 1:{
            altproctype = io;
            break;
        };
        case 2:{
            altproctype = sec;
            break;
        };
        case 3:{
            altproctype = three;
            break;
        };
        case 4:{
            altproctype = four;
            break;
        };
        case 5:{
            altproctype = five;
            break;
        };
        case 6:{
            altproctype = six;
            break;
        };
        case 7:{
            altproctype = seven;
            break;
        };
        case 8:{
            altproctype = eight;
            break;
```

```
        };
    };
    inqlist = new intlist;
    outqlist = new intlist;

    return extime;

};




//Prints a graph node
ostream& operator<<(ostream& os,gnode& g) {

    os << " ID :" << g.nodeid << "node type:" << g.ntype << endl;
    os << "AIS :" << g.aissize << " Exe time :" << g.exectime << endl;
    os << "setup time" << g.setuptime << endl;
    os << "Memory : "<<g.memid<<endl;
    os << *g.inqlist << endl;
    os << *g.outqlist << endl;
/*      os << " priority :" << g.priority << endl;
    os << "proctype :" << g.protype<<" altproctype " <<g.altproctype<<endl;*/
return os;
};

//Returns input queue list
intlist* gnode::getinputqueuelist(){
    return inqlist;
};

//Returns output queue list
intlist* gnode::getoutputqueuelist(){
    return outqlist;
};

//Returns node id
int gnode::getnodeid(){
    return nodeid;
};
```

```
//Returns node priority
int gnode::getnodepriority(){
      return priority;
};

//Returns dataperiod
int gnode::getdataperiod(){
      return dataperiod;
};

//Returns setuptime
int gnode::getsetuptime(){
      return setuptime;
};

//Returns processor type that node can run on
proctype gnode::getproctype(){
      return protype;
};

//Returns alternative processor type that node can run on
proctype gnode::getaltproctype(){
      return altproctype;
};

//Returns dataperiod
int gnode::getdatarate(){
     return dataperiod;
};

//Returns Node execuiton time
int gnode::getexectime(){
     return exectime;
};

//Increments number of queues that are already read
void gnode::incnumreadqs(){
   numreadqs++;
};
```

```
//Increments number of queues that are already written
void gnode::incnumwriteqs(){
   numwriteqs++;
};

//Returns true if last input queue of the node has been read
boolean gnode::lastnoderead(){
  if (numreadqs==totalinqs)
     return true;
  else
     return false;
};

//Returns true if the last output queue of the node has been written
boolean gnode::lastqueuewrite(){
  if (numwriteqs==totaloutqs)
     return true;
  else
     return false;
};


//Returns the fixed breakdown time
int gnode::getbreakdowntime(){
     return breakdown;
};


//Returns the node type
nodetype gnode::getnodetype(){
     return ntype;
};

boolean gnode::operator<(gnode& n){
   if (n.order> order)
     return true;
   else
     return false;
};
```

```
#include "global.h"
#include <fstream.h>
class nlist;
#ifndef QUEUEITEM_H
#define QUEUEITEM_H
class QueueItem{
 public :
  QueueItem();
  void loadqueue(fstream&,int, nlist*);
  int getgmid();
  int getthreshold();
  int getsinknode();
  int getsourcenode(){
     return nodein;
   };
  int getproductionqty();
  boolean isoverthreshold();
  boolean isthereenoughspace(){
      if ((capacity - currentlength-writeamount) >= productionqty)
        return true;
      else
        return false;
  };

  void inccurrentlength(int);

  friend ostream & operator<<(ostream&,QueueItem&);

// private :
  friend class Qlist;
  int   queueid,
        gmid,
        nodein,
        nodeout,
        thresholdqty,
        consumptionqty,
        capacity,
        currentlength,
        productionqty,
        writeamount,
        readamount;
```

```
  boolean overthreshold,
        overcapacity;

  queuetype qtype;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : GQUEUE Class Header file(Graph Queue).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>
#include "gqueue.h"
#include "nlist.h"

//Constructor
QueueItem::QueueItem() {
queueid = 0;
gmid = 1;
nodein = 0;
nodeout = 0;
thresholdqty = 0;
      currentlength=0;
capacity = 0;
      writeamount=0;
      productionqty=0;
      readamount=0;
overthreshold = false;
overcapacity = false;
      qtype = data;
};

//Loads one queue from given stream
void QueueItem::loadqueue(fstream& grphfile,int gid,nlist* nodelist){
      int temp;
grphfile  >> queueid;
      grphfile  >> temp;
      if (temp==0)
         qtype = syn_arc;
      else
         qtype = data;
grphfile  >> nodein;
      for (nodelist->current = nodelist->head;
          nodelist->current;
          nodelist->current=nodelist->current->nextitem){
          if (nodelist->current->element->getnodeid()==nodein){
```

```
            nodelist->current->element->outqlist->addtolist(queueid);
            nodelist->current->element->totaloutqs++;
          };
      };

grphfile >> nodeout;
      for (nodelist->current = nodelist->head;
          nodelist->current;
          nodelist->current=nodelist->current->nextitem){
          if (nodelist->current->element->getnodeid()==nodeout){
            nodelist->current->element->inqlist->addtolist(queueid);
            nodelist->current->element->totalinqs++;
          };
      };

grphfile >> thresholdqty;
grphfile >> currentlength;
      grphfile >> productionqty;
      consumptionqty=productionqty;
      grphfile >> capacity;
if (currentlength >= thresholdqty)
  overthreshold = true;
      gmid = gid;

};


//Returns Memory id that queue is assigned
int QueueItem::getgmid(){
   return gmid;
};

//Returns threshold Quantity
int QueueItem::getthreshold(){
   return thresholdqty;
};

//Prints a queue
ostream& operator<<(ostream& os,QueueItem& q) {

os << "ID :" << q.queueid << "NODE IN " << q.nodein << "NODEOUT " << q.nodeout << endl;
```

```
    os << "MEMORY :" << q.gmid<<endl;
    os << "THRES : " << q.thresholdqty << endl;

    os << "CAPAC : " << q.capacity << "DATARATE " << endl;
    os << "LENGTH " << q.currentlength << "overthreshold :" << q.overthreshold <<endl;

return os;
};

//Increments the current length of the queue by production quantity

void QueueItem::inccurrentlength(int t){
   currentlength=currentlength+t;
   if (currentlength >= thresholdqty)
     overthreshold = true;
   else
     overthreshold = false;
};

//Returns sink node
int  QueueItem::getsinknode(){
   return nodeout;
};

//Returns true if queue is overthreshold
boolean QueueItem::isoverthreshold(){
   if ((currentlength-readamount)>=thresholdqty)
     return true;
   else
     return false;
};

// Returns production quantity
int QueueItem::getproductionqty(){
  return productionqty;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : NLIST CLASS header file(Node List).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "gnode.h"
#include "qlist.h"
#include <fstream.h>

#ifndef NLIST_H
#define NLIST_H

class nlist{
public:
     nlist();
     ~nlist();
     gnode *getnextnode(){
          return current->nextitem->element;
     };
     int sort_topologically(Qlist*);
     gnode* getnode(int);
     int loadnodes(fstream&,fstream&,int,int);
     friend ostream& operator<<(ostream&,nlist&);


     struct nitem{
          gnode *element;
          nitem *nextitem;
     };

     nitem *head,
          *current;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : NLIST CLASS source code.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include"pnqueue.h"
#include"nlist.h"

//Constructor
nlist::nlist(){
      head = NULL;
      current= NULL;
};

//Destructor
nlist::~nlist(){
      delete head;
      delete current;
      head = NULL;
      current =NULL;
};


//Loads graph nodes
int nlist::loadnodes(fstream& grphfile,fstream& mem,int choice,int nummem){
      int numnodes = 0;
      int exectime = 0;
      int gid;
      nitem *tempn;

      grphfile >> numnodes;
      for (int loop = 1;loop <= numnodes;loop++){
          mem >> gid;
          gid = gid % nummem+1;
          tempn= new nitem;
          tempn->element = new gnode;
          if (head ==  NULL){
            exectime=exectime+tempn->element->loadnode(grphfile,choice,gid);
            current = head = tempn;
          }
```

```
        else{
          exectime=exectime+tempn->element->loadnode(grphfile,choice,gid);
          current->nextitem = tempn;
          current = current->nextitem;
        };
    };

    return exectime;
};

//Get the node whose id number is given
gnode * nlist::getnode(int nid){
      for (current=head;current;current=current->nextitem)
          if (current->element->getnodeid()==nid)
             return current->element;
       if (current == NULL)
          cerr <<nid<< "***ERROR undefined node id\n";

};


//Prints a node
ostream& operator<<(ostream& os,nlist& n){
    if (n.head == NULL)
          os << "LIST IS EMPTY\n";
    else{
          for(n.current=n.head;n.current;n.current=n.current->nextitem)
               os << *n.current->element << endl;
    };
return os;
};

int  nlist::sort_topologically(Qlist *qlst){
   int count,
     loop;
   int g_label = 1;
   int in_degrees[100];
   int nodes[100][2];
   pnqueue *q;
   intlist *ql;
   intnode *tempnode;
```

```
QueueItem *qu;
gnode *nd;
count = 0;
q = new pnqueue;
for (current=head;current;current=current->nextitem){
   nodes[count][1] = current->element->nodeid;
   cout << nodes[count][1] << "  "<< current->element->totalinqs<<endl;
   in_degrees[count] = current->element->totalinqs;
   count++;
};
count--;
for (loop =0;loop<= count;loop++)
   if (in_degrees[loop] == 0){

      tempnode = new intnode;
      tempnode->nid=nodes[loop][1];
      tempnode->priority=0;
      q->enqueue(tempnode);
   };

while (tempnode=q->dequeue()){
    for (loop=0;loop<=count;loop++)
       if (tempnode->nid==nodes[loop][1])
          nodes[loop][2]=g_label;
    nd = getnode(tempnode->nid);
    nd->order = g_label;
    ql = nd->getoutputqueuelist();
    for (ql->current=ql->head;ql->current;ql->current=ql->current->nextnode){
       qu = qlst->getqueue(ql->current->number);
       for (loop=0;loop<=count;loop++)
          if (qu->nodeout==nodes[loop][1]){
             in_degrees[loop]=in_degrees[loop]-1;
             if (in_degrees[loop]==0){
                tempnode = new intnode;
                tempnode->nid=nodes[loop][1];
                tempnode->priority=0;
                q->enqueue(tempnode);
             };
          };

    };
```

```
        g_label = g_label +1;

    };


    for (current=head;current;current=current->nextitem)
      cout << current->element->nodeid<<"--"<<current->element->order<<endl;

    return g_label - 1;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : QLIST CLASS header file(Graph Queue List).
// Date         : 12 November 1992
// Last Revised : 03 January 1993

#include "gqueue.h"

#ifndef QLIST_H
#define QLIST_H

class Qlist{
 public:
      Qlist();        //constructor
      ~Qlist();       //destructor
      QueueItem *getnextqueue(){
  return current->nextitem->element;
};
      QueueItem * getqueue(int);
      void loadqueues(fstream&,fstream&,nlist*,int);
      friend ostream& operator<<(ostream&,Qlist&);

// private :
      struct qitem{
           QueueItem *element;
           qitem *nextitem;
     };

      qitem  *head,
             *current;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : QLIST CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "qlist.h"

//constructor
Qlist::Qlist(){
    head = NULL;
    current = NULL;
}

//Destructor
Qlist::~Qlist(){
    delete head;
    delete current;
    head = NULL;
    current = NULL;
}

//Loads graph queues from file namely simdata
void Qlist::loadqueues(fstream &grphfile,fstream &mem,nlist* nolist,int nummem){
   int numqueues = 0;
   qitem *tempq;
   int gid;
   grphfile >> numqueues;
   for (int loop = 1;loop <= numqueues; loop++) {
     mem>>gid;
     gid = gid%nummem+1;
     tempq = new qitem;
     tempq->element =  new QueueItem;
if (head == NULL){
       tempq->element->loadqueue(grphfile,gid,nolist);
  current = head = tempq;
}
else{
  tempq->element->loadqueue(grphfile,gid,nolist);
  current->nextitem = tempq;
       current = current->nextitem;
```

```
};
   };
};

//Returns the graph queue whose id is given
QueueItem* Qlist::getqueue(int quid){
      for (current=head;current;current=current->nextitem)
         if (current->element->queueid == quid)
           return current->element;
      if (current == NULL)
         cerr << "****ERROR NO SUCH QUEUE\n";
};

//Prints the graph queue list
ostream& operator<<(ostream& os,Qlist& q){
if (q.head == NULL)
  os << "QUEUE IS EMPTY" << endl;
      else{
  for(q.current=q.head;q.current;q.current=q.current->nextitem)
    os << *q.current->element << endl;
}
      return os;
}
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : MEMORY Class Header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "global.h"


class memory{
public:
     memory();
     void setbusytill(double);
     void setbusy(boolean);
     boolean isbusy();
     int getmemid();
     int getbusytime();
     void setobjectid(int);

private:
     double     busytill;
     int        memid,
                capacity,
                granularity,
                setuptime,
                bandwith;

     status_type  status;
     boolean     busy;

};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : LGDF machine simulator class.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include<iostream.h>
#include"memory.h"

//Constructor
memory::memory(){
    memid = 0;
    busytill = 0;
    capacity = 0;
    granularity = 0;
    setuptime = 0;
    bandwith = 0;
    status = active;
    busy = false;
};

//Updates the busy time if it is in the future
void memory::setbusytill(double t){
   busytill = t;
};

//Set the given boolean value
void memory::setbusy(boolean value){
   busy = value;
};


//Returns true if the memory is busy
boolean memory::isbusy(){
   return busy;
};

//Returns memory id
int memory::getmemid(){
   return memid;
};
```

```
//Returns the time that memory will stay busy
int memory::getbusytime(){
    return busytill;
};

//Sets the given number as object id
void memory::setobjectid(int t){
    memid = t;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : MLIST CLASS(Memory list).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include"memory.h"
#include<fstream.h>

class mlist {
public:
      mlist();
      ~mlist();
      void addtolist(memory *);
      memory* getmem(int);
      int loadmemory(fstream&);

private:
      struct  memnode{
            memory* m;
            memnode* nextmem;
      };

      memnode* head,
            * current;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : MLIST CLASS source code.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>
#include "mlist.h"
extern "C"{
  exit(int);
}

//Constructor
mlist::mlist(){
     head = NULL;
     current = NULL;
};

//Destructor
mlist::~mlist(){
     delete head;
     delete current;
     head = NULL;
     current = NULL;
};

//Adds given memory to the list
void mlist::addtolist(memory * mem){
     if (head == NULL){
       head = new memnode;
       head->m = mem;
       current = head;
     }
     else {
       for (current=head;current->nextmem;current=current->nextmem);
       current->nextmem = new memnode;
       current->nextmem->m = mem;
     };

};
```

```
//Returns the memory whose id is given
memory * mlist::getmem(int mid){
        for (current=head;current;current=current->nextmem)
           if (current->m->getmemid()==mid)
              return current->m;
         cout << mid << " there is no such memory";
         if (current == NULL){
           exit(1);
           cerr << "***ERROR undefined memory id\n";
         };
};

//Loads memories
int  mlist::loadmemory(fstream& grphfile){
         int nummemory=0;
         memnode* temp;
         grphfile >> nummemory;
         for (int loop=1;loop<=nummemory;loop++){
            temp=new memnode;
            temp->m=new memory;
            temp->m->setobjectid(loop);
            if (head == NULL)
              current = head =temp;
            else{
              current->nextmem=temp;
              current=current->nextmem;
            };
          };
cout <<nummemory;
          return nummemory;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PROCESSOR CLASS header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include "global.h"

#ifndef PROCESSOR_H
#define PROCESSOR_H

class processor{
public:
     processor();
     void setsetupbusytill(double);
     void setexecbusytill(double);
     void setbreakdownbusytill(double);
     void setsetupbusy(boolean);
     void setexecbusy(boolean);
     void setbreakdownbusy(boolean);
     void setfree(boolean);
     void setproctype(int);
     void setobjectid(int);
     boolean isfree();
     boolean issetupbusy();
     boolean isexecbusy();
     boolean isbreakdownbusy();
     proctype gettype();
     double getsetupbusytime();
     double getbreakdownbusytime();
     double getexecbusytime();

     int getprocid();


//private:

     proctype  ptype;

     double    setupbusytill,
```

```
            execbusytill,
            breakdownbusytill,
            durationprocbusy,
            onlyexectime,
            startutil;


    int procid,
      speed;
    status_type status;
    boolean readinginprocess,
            writeinprocess,
            waitingonsetup,
            waitingonexec,
            free,
            setupbusy,
            execbusy,
            breakdownbusy,
            pendingsetup,
            pendingbreakdown;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PROCESSOR CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include<iostream.h>
#include"processor.h"

//Constructor
processor::processor(){
    ptype = inst;
    procid = 0;
    setupbusytill = 0;
    execbusytill = 0;
    breakdownbusytill =0;
    durationprocbusy=0;
    startutil = 0;
    setupbusy = false;
    execbusy = false;
    readinginprocess=false;
    writeinprocess=false;
    setupbusy=false;
    execbusy = false;
    breakdownbusy = false;
    pendingsetup = false;
    pendingbreakdown = false;
    speed = 0;
    status =  active;
    free = true;
    waitingonsetup = false;
    waitingonexec = false;
};


//Updates the busy time of the setup stage
void processor::setsetupbusytill(double t){
    if (setupbusytill < t)
       setupbusytill = t;
};
```

```
//Updates the busy time of execution stage
void processor::setexecbusytill(double t){
   if (execbusytill < t)
      execbusytill = t;
};


//Updates the breakdown stage busy time
void processor::setbreakdownbusytill(double t){
  if (breakdownbusytill < t)
     breakdownbusytill = t;
};


//Sets setup busy flag to the given boolean value
void processor::setsetupbusy(boolean value){
   setupbusy = value;
};

//Sets execution stage flag to the given boolean value
void processor::setexecbusy(boolean value){
   execbusy = value;
};

//Sets breakdown busy flag to the given boolean value
void processor::setbreakdownbusy(boolean value){
   breakdownbusy = value;
};


//Sets the processor free flag to the given boolean value
void processor::setfree(boolean value){
   free = value;
};

//Returns true if the processor is free
boolean processor::isfree(){
   return free;
};
```

```
//Returns true if setup stage is busy
boolean processor::issetupbusy(){
   return setupbusy;
};

//Returns true if the execution stage is busy
boolean processor::isexecbusy(){
   return execbusy;
};

//Returns true if breakdown stage is busy
boolean processor::isbreakdownbusy(){
   return breakdownbusy;
};

//Returns the processor id
int processor::getprocid(){
   return procid;
};

//Returns time that setup stage will stay busy
double processor::getsetupbusytime(){
   return setupbusytill;
};

//Returns time that execution stage will stay busy
double processor::getexecbusytime(){
  return execbusytill;
};

//Returns time that breakdown stage will stay busy
double processor::getbreakdownbusytime(){
  return breakdownbusytill;
};

//Returns processor type
proctype processor::gettype(){
  return ptype;
};

//Sets the object id to the given value
```

```
void processor::setobjectid(int t){
   procid = t;
};

//Sets the processor type according to given integer value
void processor::setproctype(int t){
  switch (t){
    case 0:{
        ptype = inst;
        break;
    };
    case 1:{
        ptype = io;
        break;
    };
    case 2:{
        ptype = sec;
        break;
    };
    case 3:{
          ptype = three;
          break;
    };
    case 4:{
          ptype = four;
          break;
    };
    case 5:{
          ptype = five;
          break;
    };
    case 6:{
          ptype = six;
          break;
    };
    case 7:{
          ptype = seven;
          break;
    };
    case 8:{
          ptype = eight;
```

```
            break;
        };

    };
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PLIST CLASS header file(Processor list).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include"processor.h"


#include<fstream.h>
#ifndef PLIST_H
#define PLIST_H
class plist {
public:
      plist();
      ~plist();
      void addtolist(processor*);
      processor* getproc(int);
      int loadprocessors(fstream&,int);
//private:
      struct  procnode{
            processor* p;
            procnode* nextproc;
      };

      procnode* head,
           * current;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PLIST CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>
#include "plist.h"
#include <fstream.h>

//Constructor
plist::plist(){
    head = NULL;
    current = NULL;
};

//Destructor
plist::~plist(){
    delete head;
    delete current;
    head = NULL;
    current = NULL;
};

//Adds given processor to processor list
void plist::addtolist(processor * proc){
    if (head == NULL){
      head = new procnode;
      head->p = proc;
      current = head;
    }
    else {
      for (current=head;current->nextproc;current=current->nextproc);
      current->nextproc = new procnode;
      current->nextproc->p = proc;
    };

};

//Returns the processor whose id is given
processor* plist::getproc(int pid){
```

```
        for (current=head;current;current=current->nextproc)
            if (current->p->getprocid()==pid)
                return current->p;
        if (current == NULL) {
            cout <<"***ERROR undefined processor id\n";
        }
};

//Loads processor from given stream
int  plist::loadprocessors(fstream& grphfile,int numprocs){
    int    prtype;
    int proccnt = 0;

    procnode* temp;
    for (int loop=1;loop <= numprocs;loop++) {
        temp = new procnode;
        temp->p= new processor;
        temp->p->setobjectid(loop);
        grphfile >> prtype;
        if (prtype != 1)
            proccnt++;
        temp->p->setproctype(prtype);
        if (head ==NULL)
            current=head=temp;
        else{
            current->nextproc=temp;
            current=current->nextproc;
        };
    };
    return proccnt;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : SCHEDULER CLASS header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "pnqueue.h"
#include "gnode.h"
#include "nlist.h"
#include "plist.h"
#include "pqueue.h"
class scheduler {
public :
    scheduler();
    ~scheduler();
    void putnodeinrl(gnode* );
    void putprocinfl(int);
    void printreadylist();
    void printproclist();
    boolean isbusy();
    void setbusy(boolean);
    int getbusytill();
    int getnodefromrl();
    void setbusytill(int);
    void schedule_node(nlist*,plist*,pqueue*,double);
    boolean emptyrl();
    boolean member(int);
//private:
    policy_type  policy;

    double       processorid,
                 busytill,
                 schedulingtime;

    boolean     busy;

    pnqueue*    readynodelist,
          *     freeproclist;

};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : SCHEDULER CLASS source code.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>
#include "scheduler.h"
#include "global.h"
#include "event.h"


//Constructor
scheduler::scheduler(){
      policy = synhronized;
      processorid = 0;
      schedulingtime = 0;
      readynodelist = new pnqueue;
      freeproclist  = new pnqueue;
};

//Destructor
scheduler::~scheduler(){
      delete readynodelist;
      readynodelist=NULL;
};

//Puts the given graph node to the ready node list
void scheduler::putnodeinrl(gnode* n){
      intnode* tempnode=new  intnode;
//       n->priority = n->nodeinstance*(-1);
      tempnode->setnode(n->getnodeid(),n->getnodepriority());
      readynodelist->enqueue(tempnode);
};

//Puts the given processor to the free processor list
void scheduler::putprocinfl(int pid){
      intnode* tempnode=new  intnode;
      tempnode->setnode(pid,0);
      freeproclist->enqueue(tempnode);
};
```

```
//Sets scheduler busy flay to the given boolean value
void scheduler::setbusy(boolean value){
      busy = value;
};

//Returns true if the scheduler is busy
boolean scheduler::isbusy(){
      return busy;
};

//Returns the time that scheduler will be busy
int scheduler::getbusytill(){
      return busytill;
};

//Returns a node from ready node list if queue is empty return -1
int scheduler::getnodefromrl(){
     intnode* temp;
     temp=readynodelist->dequeue();
     if (temp !=NULL)
        return temp->geteid();
     else
        return -1;
};

//Updates the scheduler busy time to the given value
void scheduler::setbusytill(int t){
    busytill=t;
};


//Returns true if the ready node list is empty
boolean scheduler::emptyrl(){
    if (readynodelist->head == NULL)
      return true;
    else
      return false;
};

//Schedules a node from ready list by matching it to a free processor and
```

```
// produces start setup event.First it tries to  find a node which is not
//currently executing
void scheduler::schedule_node(nlist* nl,plist *pl,pqueue* eventqueue,double clck){

  event* tempevent= new event;
  gnode* n;
  processor *p;
  pnqueue::item* previous,
          * previous2;



  if (readynodelist->head != NULL){
   previous = NULL;
   for (readynodelist->current=readynodelist->head;
     readynodelist->current;
     readynodelist->current= readynodelist->current->nextitem){
     n = nl->getnode(readynodelist->current->nodeitem->geteid());
     if (!(n->processing)){
      previous2 = NULL;

      for (freeproclist->current=freeproclist->head;
         freeproclist->current;
         freeproclist->current=freeproclist->current->nextitem){

          p = pl->getproc(freeproclist->current->nodeitem->geteid());
          if((n->getproctype()==p->gettype()) ||
            (n->getaltproctype() == p->gettype() )){
                busy = true;
                busytill = clck+schedulingtime;
                tempevent->eventname=free_scheduler;
                tempevent->starttime=busytill;
                tempevent->priority =busytill;
                eventqueue->enqueue(tempevent);

               if (previous2 == NULL)
                freeproclist->head=freeproclist->current->nextitem;
               else
                 previous2->nextitem=freeproclist->current->nextitem;
```

```
            if (previous == NULL)
              readynodelist->head=readynodelist->current->nextitem;
            else
              previous->nextitem=readynodelist->current->nextitem;
              p->setfree(false);
              n->processing= true;
              n->exectimes++;
              n->nodeinstance++;
              tempevent= new event;
              tempevent->eventname=start_setup;
              tempevent->starttime=clck;
              tempevent->priority =clck;
              tempevent->nodenum  =n->getnodeid();
              tempevent->assocproc=p->getprocid();
              eventqueue->enqueue(tempevent);
/*          if (p->gettype()!=io){
              cout << "PROCESSOR ID :"<<tempevent->assocproc;
              cout << "  NODE ID     :"<<tempevent->nodenum<<endl;
            };*/
              return ;

      };
      previous2 = freeproclist->current;
    };

/*

    for (pl->current=pl->head;pl->current;pl->current=pl->current->nextproc){

      if((n->getproctype()==pl->current->p->gettype()) ||
        (n->getaltproctype() == pl->current->p->gettype() )){
        if (pl->current->p->isfree()){
            busy = true;
            busytill = clck+schedulingtime;
            tempevent->eventname=free_scheduler;
            tempevent->starttime=busytill;
            tempevent->priority =busytill;
            eventqueue->enqueue(tempevent);

          if (previous == NULL)
            readynodelist->head=readynodelist->current->nextitem;
```

```
                else
                  previous->nextitem=readynodelist->current->nextitem;


                  pl->current->p->setfree(false);
                  n->processing= true;
                  n->exectimes++;
                  n->nodeinstance++;
                  tempevent= new event;
                  tempevent->eventname=start_setup;
                  tempevent->starttime=clck;
                  tempevent->priority =clck;
                  tempevent->nodenum  =n->getnodeid();
                  tempevent->assocproc=pl->current->p->getprocid();
                  eventqueue->enqueue(tempevent);
                  return ;
              };
          };
        };
*/


      };
      previous = readynodelist->current;
    };
  };
};

//Returns true if the given node is already in the ready node list
boolean scheduler::member(int num){
   if (readynodelist->head != NULL)
    for (readynodelist->current=readynodelist->head;
      readynodelist->current;
      readynodelist->current= readynodelist->current->nextitem)
      if (readynodelist->current->nodeitem->geteid()==num)
         return true;

   return false;
};
```

```
//Prints the ready node list
void scheduler::printreadylist(){
    cout << "<<<<<< ";
    for (readynodelist->current=readynodelist->head;
      readynodelist->current;
      readynodelist->current= readynodelist->current->nextitem)
      cout << readynodelist->current->nodeitem->nid << "  ";

    cout << ">>>>>>\n";
};

//Prints the free processor list
void scheduler::printproclist(){
    cout << "<<<<<< ";
    for (freeproclist->current=freeproclist->head;
      freeproclist->current;
      freeproclist->current= freeproclist->current->nextitem)
      cout << freeproclist->current->nodeitem->nid << "  ";

    cout << ">>>>>>\n";
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PNQUEUE CLASS header file (Priority Queue For integer node.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "node.h"
#ifndef PNQUEUE_H
#define PNQUEUE_H

class pnqueue{

public:
      pnqueue();
      ~pnqueue();
      void enqueue(intnode*);
      intnode* dequeue();

//private:
      struct item{
      intnode *nodeitem;
      item *nextitem;
       };

      item  *head,
          *current;

};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PNQUEUE CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "pnqueue.h"
#include <iostream.h>

//Constructor
pnqueue::pnqueue(){
      head = NULL;
      current = NULL;
 };

//Destructor
pnqueue :: ~pnqueue(){
      delete head;
      delete current;
      head = NULL;
      current = NULL;
};

//Enqueues the given integer node to queue
void pnqueue :: enqueue(intnode  *e)  {
       item *tempnode=new item;
       item *previous=head;
  tempnode->nodeitem=e;

       if (head == NULL)
           current=head=tempnode;
       else{
           current= head;
           while ((current->nextitem!=NULL) &&
             (current->nodeitem->getpriority()<=e->getpriority())){
               previous = current;
               current = current->nextitem;
 };

           if ((current->nodeitem->getpriority() > e->getpriority())&&
             (current != NULL)&&(current!=head)){
```

```
previous->nextitem=tempnode;
tempnode->nextitem=current;
}
else{
        if((current->nextitem==NULL) &&
                (current->nodeitem->getpriority() <= e->getpriority())){
                  current->nextitem =  tempnode;
  }
              else{
                  current = head;
                  head = tempnode;
                  head->nextitem = current;
               };
            };
       };
};


//Returns integer node from queue
intnode * pnqueue :: dequeue(){
     item* temp;
     temp= head;
     if (head != NULL){
        head=head->nextitem;
        current = head;
        return temp->nodeitem;
     };

 };
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PQUEUE CLASS header file(Priority queue for events).
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include "event.h"
#ifndef PQUEUE_H
#define PQUEUE_H
class pqueue{

public:
      pqueue();
      ~pqueue();
      void enqueue(event*);
      void printeq();
      event* dequeue();
      boolean empty();

private:
      struct it{
      event *eventitem;
      it *nextitem;
       };

      it  *head,
          *current;

};
#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : PQUEUE CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include "pqueue.h"
#include <iostream.h>

//constructor
pqueue::pqueue(){
      head = NULL;
      current = NULL;
 };

//Destructor
pqueue :: ~pqueue(){
      delete head;
      delete current;
      head = NULL;
      current = NULL;
 };

//Enqueues the given event according its priority
void pqueue :: enqueue(event  *e)  {
       it *tempnode=new it;
       it *previous=head;
  tempnode->eventitem=e;

       if (head == NULL)
           current=head=tempnode;
       else{
           current= head;
           while ((current->nextitem!=NULL) &&
             (current->eventitem->getpriority()<=e->getpriority())){
               previous = current;
               current = current->nextitem;
 };

           if ((current->eventitem->getpriority() > e->getpriority())&&
```

```
            (current != NULL)&&(current!=head)){
previous->nextitem=tempnode;
tempnode->nextitem=current;
}
else{
        if((current->nextitem==NULL) &&
              (current->eventitem->getpriority() <= e->getpriority())){
                current->nextitem =  tempnode;
  }
            else{
                current = head;
                head = tempnode;
                head->nextitem = current;
             };
          };
       };
};
//Dequeues the event from the top of the queue
event * pqueue :: dequeue(){
    it* temp;
    temp= head;
    head=head->nextitem;
    current = head;
    return temp->eventitem;
};

//Returns true if the Queue is empty
boolean pqueue:: empty(){
    if (head==NULL)
      return true;
    else
      return false;
};

//Prints the queue
void pqueue::printeq(){
    cout << "< ";
    for (current = head;current;current = current->nextitem)
       cout << current->eventitem->geteventname() << ' ';
    cout << "> \n";
};
```

```
// Author        : Cem Akin
// Advisor       : Amr Zaky
// Description   : NODE CLASS header file.
// Date          : 12 November 1992
// Last Revised  : 03 January  1993

#include <iostream.h>

#ifndef NODE_H
#define NODE_H
class intnode{
 public:
      intnode();
int getpriority();
      int geteid();
      void setnode(int ,int);
friend ostream& operator<<(ostream&,intnode&);
// private:
      int nid,
   priority;
};
#endif;
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : NODE CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "node.h"

//Constructor
intnode::intnode(){
nid =0;
priority = 0;
};

//Returns node priority
int intnode::getpriority(){
return priority;
};

//Returns node id
int intnode::geteid(){
return nid;
};

//Set node with id and prty
void intnode::setnode(int id,int prty){
nid = id;
priority = prty;

};

//Prints a node
ostream& operator<<(ostream& os, intnode& e){
os << "Event ID: "<<e.nid;
      os << endl;
      os << "priority :" << e.priority;
      os << endl;
      return os;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : Integer List Class header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "global.h"
#ifndef ILIST_H
#define ILIST_H

class intlist{
public:
      intlist();
      ~intlist();
      void addtolist(int);
      boolean finditem(int );
      friend ostream& operator<<(ostream&,intlist&);

//private:
      struct intnode{
         int number;
         intnode *nextnode;
      };
      intnode *head,
           *current;

};
#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : Integer List Class source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include<iostream.h>
#include "ilist.h"

//Contructor
intlist::intlist(){
      head = NULL;
      current = NULL;
};

//Destructor
intlist::~intlist(){
      delete head;
      delete current;
      head = NULL;
      current = NULL;
};

//If given integer is in list returns true
boolean intlist::finditem(int num){
boolean result=false;
if (head ==NULL)
        return result;
      else
   for(current=head;current->nextnode;current = current->nextnode)
           if(current->number ==  num)
result=true;
return result;
};

//Adds the given integer to the list
void intlist::addtolist(int num){
      if (head==NULL){
           head = new intnode;
           head->number = num;
           current=head;
```

```
    }
    else{
        for(current=head;current->nextnode;current=current->nextnode);
        current->nextnode=new intnode;
        current->nextnode->number=num;
        current = current->nextnode;
    };
};

//Prints the integer list
ostream& operator<<(ostream& os,intlist& ilist) {
    os << '<' << " ";
    for (ilist.current = ilist.head;ilist.current;
            ilist.current = ilist.current->nextnode)
        os << ilist.current->number << " ";
    os << '>' << endl;
    return os;
};
```

APPENDIX C: Graph Restructure Source Code

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : CNODE CLASS header file.
// Date         : 10 March 1993
// Last Revised  : 13 March  1993


class cnode{//This class is used to represent a space that is
        //assigned to a graph node.

 public :
    cnode(){
      id =0;
      start = 0;
      finish = 0;
    };
    cnode(int a){
      id =0;
      start = 0;
      finish = a;
    };


    int id;
    int start,
        finish;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : CYLINDER CLASS header file.
// Date         : 10 March 1993
// Last Revised  : 13 March  1993


#include "nodelist.h"
#include "nlist.h"
#include "qlist.h"



class cylinder{//This class is used to represent a cylinder
 public:
      cylinder();
      ~cylinder();
      void initialize_cylinder();
      void cylinder_assignment();
      void cylinder::initialize_times();
      int find_latest_end_parent(int);
      boolean  map_cylinder();



      struct cyl_slice{
          nodelist  *slice;
          nodelist  *el;
      };

      cyl_slice cylin[20];
      int circum;
      int ps;
      nlist *gnodelist,
          *sortedlist;
      Qlist *gqueuelist;
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : Cylinder Class Source file.
// Date         : 10 March 1993
// Last Revised : 13 March 1993


#include "cylinder.h"
#include <iostream.h>
    //Cylinder class constructor
    cylinder::cylinder(){
        int c;
        int ps = 0;
        int circum = 0;
        gnodelist = new nlist;
        sortedlist=new nlist;
        gqueuelist = new Qlist;
        for (c =0;c<20;c++){
          cylin[c].slice = NULL;
          cylin[c].el = NULL;
        };
    };
    //Cylinder class destructor
    cylinder::~cylinder(){
        int c;
        delete gnodelist;
        delete gqueuelist;
        for (c=0;c<20;c++){
          delete cylin[c].slice;
          delete cylin[c].el;
          cylin[c].slice = NULL;
          cylin[c].el = NULL;
        };
        gnodelist = NULL;
        gqueuelist = NULL;
    };
    //Initialize the cylinder to empty cylinder
    void cylinder::initialize_cylinder(){
        int c;
        for (c=0;c<20;c++){
          delete cylin[c].slice;
```

```
            delete cylin[c].el;
            cylin[c].slice = new nodelist;
            cylin[c].el = new nodelist(circum);
        };
};
//Clears the assigned execution times when mapping attemp is not
//succesfull
void cylinder::initialize_times(){
    for (gnodelist->current=gnodelist->head;
        gnodelist->current;
        gnodelist->current=gnodelist->current->nextitem){
        gnodelist->current->element->start=0;
        gnodelist->current->element->finish=0;
    };
};
//Finds the latest end parent for a given node and returns it
int cylinder::find_latest_end_parent(int num){
    int  max=0;
    gnode* tempnode,
         * tempnode2;
    tempnode=gnodelist->getnode(num);
    for(tempnode->parentlist->current=tempnode->parentlist->head;
        tempnode->parentlist->current;
        tempnode->parentlist->current=tempnode->parentlist->current->nextnode){
        tempnode2=gnodelist->getnode(tempnode->parentlist->current->number);
        if (max<tempnode2->finish)
            max = int(tempnode2->finish);
    };
    return max;
};

//Maps the cylinder according to its topology
void cylinder::cylinder_assignment (){
        boolean flag;
        fstream cyl;
        double  percent;
        int     c,numnodes;
        c = gnodelist->sort_topologically(gqueuelist,sortedlist);
        gnodelist = sortedlist;
        flag = map_cylinder();
        while (!flag){
```

```
        cout << "I COULD NOT FIND ANY SOLUTION WITH ";
        cout << circum << ".\n";
        cout << "I NEED LARGER CIRCUMFERENCE.WILL YOU GIVE ME A \n";
       cout << "PERCENT THAT I CAN INCREASE THE SIZE OF CYLINDER\n";
        cout << "PERCENT  : ";
        cin  >> percent;
        cout << "\nTHANK YOU NOW I AM TRYING TO FIND A SOLUTION\n";
         circum = int(circum + circum * percent);
         initialize_cylinder();
         initialize_times();
         flag=map_cylinder();
      };
      cyl.open("cyl.dat",ios::out);
      for (c=0;c<ps;c++){
         numnodes = 0;
         for (cylin[c].slice->current=cylin[c].slice->head;
            cylin[c].slice->current;
            cylin[c].slice->current=cylin[c].slice->current->nextitem){
            numnodes++;
            cout << cylin[c].slice->current->element->id<<" ";
            cout << cylin[c].slice->current->element->start<<"  ";
            cout << cylin[c].slice->current->element->finish<<endl;
         };
         cout << endl;
         cout <<"**********************************************\n";
         cyl << numnodes<<endl;
         for (cylin[c].slice->current=cylin[c].slice->head;
            cylin[c].slice->current;
            cylin[c].slice->current=cylin[c].slice->current->nextitem){
            cyl << cylin[c].slice->current->element->id<<" ";
         };
         cyl <<endl;
      };
      cout << circum<<endl;
      cyl  << circum<<endl;
};

//Tries to find a solution for mapping and called by cylider assignment
//function
boolean  cylinder::map_cylinder(){
      nitem *tptr;
```

```
cnode *tempspace,
    *tempcnode;
boolean found1 = false;
boolean found2 = false;
boolean found3 = false;

int  minstart1 = circum;
int  minstart2 = circum;
int  minstart3 = circum;

int  t,
     t1,
     t2,
     t3,
     t4,
     t5,
     t6;
int cnt,s1,s2,s3,sid,c;
int count=0;

for (gnodelist->current = gnodelist->head;
   gnodelist->current;
   gnodelist->current = gnodelist->current->nextitem){
 if (gnodelist->current->element->ntype==instruction){
   tptr = gnodelist->current;
   t = find_latest_end_parent(gnodelist->current->element->nodeid);
   gnodelist->current=tptr;
   for (cnt=0;cnt<ps;cnt++)
     for (cylin[cnt].el->current=cylin[cnt].el->head;
        cylin[cnt].el->current;
        cylin[cnt].el->current=cylin[cnt].el->current->nextitem){
        if ((cylin[cnt].el->current->element->start>=t)&&
           (cylin[cnt].el->current->element->finish >=
            cylin[cnt].el->current->element->start +
           gnodelist->current->element->exectime))
           if (cylin[cnt].el->current->element->start<minstart1){
             minstart1 = cylin[cnt].el->current->element->start;
             found1 = true;
             sid = cylin[cnt].el->current->element->id;
             t1 = cylin[cnt].el->current->element->start;
             t2 = cylin[cnt].el->current->element->finish;
```

```
            s1 = cnt;
          };
        if ((cylin[cnt].el->current->element->start<t)&&
          (cylin[cnt].el->current->element->finish >=t +
          gnodelist->current->element->exectime))
          if (cylin[cnt].el->current->element->start<minstart3){
            minstart3 = cylin[cnt].el->current->element->start;
            found3 = true;
            sid = cylin[cnt].el->current->element->id;
            t5 = cylin[cnt].el->current->element->start;
            t6 = cylin[cnt].el->current->element->finish;
            s3 = cnt;
          };
        if ((cylin[cnt].el->current->element->start+
          gnodelist->current->element->exectime) <=
          cylin[cnt].el->current->element->finish)
          if (cylin[cnt].el->current->element->start<minstart2){
            minstart2=cylin[cnt].el->current->element->start;
            sid = cylin[cnt].el->current->element->id;
            s2 = cnt;
            found2 = true;
            t3 = cylin[cnt].el->current->element->start;
            t4 = cylin[cnt].el->current->element->finish;
          };


};

if (found1){
  tempcnode = new cnode;
  tempcnode->id = gnodelist->current->element->nodeid;
  tempcnode->start = t1;
  tempcnode->finish = t1 + gnodelist->current->element->exectime;
  gnodelist->current->element->start=t1;
  gnodelist->current->element->finish=tempcnode->finish;
  cylin[s1].slice->insert(tempcnode);
  cylin[s1].el->remove(t1);
  if (t1<t){
    tempspace = new cnode;
    tempspace->id = ++count;
    tempspace->start = t1;
```

```
            tempspace->finish= t;
            cylin[s1].el->insert(tempspace);
          };
          if (tempcnode->finish < t2){
            tempspace = new cnode;
            tempspace->id = ++count;
            tempspace->start =  tempcnode->finish;
            tempspace->finish=t2;
            cylin[s1].el->insert(tempspace);

          };
/*          for (cylin[s1].el->current=cylin[s1].el->head;
             cylin[s1].el->current;
             cylin[s1].el->current=cylin[s1].el->current->nextitem){
          cout << cylin[s1].el->current->element->start << " ";
          cout << cylin[s1].el->current->element->finish<<endl;
};
*/
          found1 = false;
          found2 = false;
          found3 = false;
          minstart1 = circum;
          minstart2 = circum;
          minstart3 = circum;
        }
        else
          if (found3){
            tempcnode = new cnode;
            tempcnode->id = gnodelist->current->element->nodeid;
            tempcnode->start = t;
            tempcnode->finish = t + gnodelist->current->element->exectime;
            gnodelist->current->element->start=t;
            gnodelist->current->element->finish=tempcnode->finish;
            cylin[s3].slice->insert(tempcnode);
            if (tempcnode->id==26)
              cout << t <<" "<<t5<<" "<<t6<<endl;
            cylin[s3].el->remove(t5);
            if (t5<t){
              tempspace = new cnode;
              tempspace->id = ++count;
              tempspace->start = t5;
```

```
    tempspace->finish= t;
    cylin[s3].el->insert(tempspace);
   };
   if (tempcnode->finish < t6){
    tempspace = new cnode;
    tempspace->id = ++count;
    tempspace->start =  tempcnode->finish;
    tempspace->finish=t6;
    cylin[s3].el->insert(tempspace);
   };
   found1 = false;
   found2 = false;
   found3 = false;
   minstart1 = circum;
   minstart2 = circum;
   minstart3 = circum;
}
else
 if (found2){
   tempcnode = new cnode;
   tempcnode->id = gnodelist->current->element->nodeid;
   tempcnode->start = t3;
   tempcnode->finish = t3 + gnodelist->current->element->exectime;
   gnodelist->current->element->start=t1;
   gnodelist->current->element->finish=tempcnode->finish;
  if (tempcnode->id==26)
    cout << t <<" "<<t3<<" "<<t4<<endl;
   cylin[s2].slice->insert(tempcnode);
   cylin[s2].el->remove(t3);
   if (tempcnode->finish < t4){
    tempspace = new cnode;
    tempspace->id = ++count;
    tempspace->start =  tempcnode->finish;
    tempspace->finish=t4;
    cylin[s2].el->insert(tempspace);
   };
   found1 = false;
   found2 = false;
   found3 = false;
   minstart1 = circum;
   minstart2 = circum;
```

```
                minstart3 = circum;
              }
              else
               return false;


          };
        };
        cout << "CYLINDER FRAGMANTATION \n";
        for (c=0;c<ps;c++){
         for (cylin[c].el->current=cylin[c].el->head;
             cylin[c].el->current;
             cylin[c].el->current=cylin[c].el->current->nextitem){
          cout << cylin[c].el->current->element->id<<"  ";
          cout << "start....: "<<cylin[c].el->current->element->start;
          cout << "finish....: "<<cylin[c].el->current->element->finish;
          cout << endl;
          };
          cout<<"*****************************************************\n";
        };
    return true;
};

main(){
   int  tottime;
   fstream myfile;
   myfile.open("simdata",ios::in);
   cylinder *c = new cylinder;
   cout << "Enter Cylinder Circumference  :";
   cin >> c->circum;
   cout << c->circum;
   cout << "\nEnter Processor Number      :";
   cin >> c->ps;
   c->initialize_cylinder();
   tottime = c->gnodelist->loadnodes(myfile);
   c->gqueuelist->loadqueues(myfile,c->gnodelist);
   c->cylinder_assignment();
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : ARCNODE CLASS header file.
// Date         : 12 November 1992
// Last Revised : 03 January 1993


class arcnode{
 public:
    int sourcenodeid,
       sinknodeid,
       initial_length,
       threshold,
       production,
       consumption,
       capacity;
    arcnode(){
       sourcenodeid = 0;
       sinknodeid  = 0;
       initial_length=0;
       threshold    =0;
       production   =0;
       consumption  =0;
       capacity     =0;
};

};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : ARCNODE CLASS header file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include "arcnode.h"
#include "global.h"
#ifndef ARCLIST_H
#define ARCLIST_H


class arclist{
   public:
     arclist();        //constructor
     ~arclist();       //destructor
     boolean is_already_exist(int, int);
     void addtolist(arcnode*);

// private :
     struct sitem{
          arcnode *element;
          sitem   *nextitem;
         };

     sitem  *head,
            *current;
};

#endif
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : ARCLIST CLASS source file.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993


#include <iostream.h>
#include "arclist.h"

//Constructor
arclist::arclist(){
      head = NULL;
      current = NULL;
};

//Destructor
arclist::~arclist(){
      delete head;
      delete current;
      head = NULL;
      current = NULL;
};

boolean arclist::is_already_exist(int nr,int ns){
      for (current=head;current;current=current->nextitem)
         if ((current->element->sourcenodeid==ns)&&
            (current->element->sinknodeid==nr))
           return true;

      return false;
};


void arclist::addtolist(arcnode* an){
      sitem* tempn;
      tempn = new sitem;
      tempn->element=an;

      if (head==NULL)
        head = current = tempn;
```

```
    else{
      for (current=head;current->nextitem;current=current->nextitem);
      current->nextitem=tempn;
      current=tempn;
    };
};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : CYLINDER CLASS header file.This used for index assignment and dependency
//                arc creation
// Date         : 12 November 1992
// Last Revised  : 03 January  1993

#include "clist.h"
#include "arclist.h"
class cylinder {
 public:
    struct slice{
        int slicenum;
        clist*  nodelist;
        slice *nextslice;
    };

    cylinder(){
        head = NULL;
        current = NULL;
};

    ~cylinder(){
        delete head;
        delete current;
        head = NULL;
        current = NULL;
};
    cnode* getnode(int);
    cnode* find_latest(double);
    cnode* find_latest_start(double);
    void find_RC_arcs(int);
    void start_after_start(int);
    void start_after_finish(int);
    void assign_indices(int,int);
   slice *head,
        *current;

};
```

```
// Author       : Cem Akin
// Advisor      : Amr Zaky
// Description  : CYLINDER CLASS source file.This used for index assignment and dependency
//                arc creation.
// Date         : 12 November 1992
// Last Revised  : 03 January  1993
#include "cylinder.h"
#include "nlist.h"
#include "qlist.h"
#include "mlist.h"
#include "plist.h"

nlist *gnodelist;
Qlist *gqueuelist;

int numprocs,
   numqueues,
   numnodes,
   nummemory,
   cyl_circum;

cnode* cylinder::getnode(int nid){
  for (current=head;current;current=current->nextslice)
    for(current->nodelist->current=current->nodelist->head;
       current->nodelist->current;
       current->nodelist->current=current->nodelist->current->nextitem)
       if (current->nodelist->current->element->nodeid==nid)
          return current->nodelist->current->element;
  return NULL;
};

void cylinder::assign_indices(int nid,int index){

   gnode *tq1,
        *tq2;
   cnode *tcq1,
        *tcq2;

   QueueItem *tempqueue;
   intlist *inputqslst,
         *inqs,
```

```
            *outqs,
            *outputqslst;

      inputqslst=new intlist;
        outputqslst= new intlist;


        tq1 = gnodelist->getnode(nid);
        if (tq1->ntype==instruction){
        tcq1= getnode(nid);
        cout << nid <<"....."<<index<<endl;
        tcq1->index = index;
        tcq1->indexed = true;
        inqs=tq1->getinputqueuelist();
        outqs=tq1->getoutputqueuelist();
        for (inqs->current=inqs->head;
            inqs->current;
            inqs->current=inqs->current->nextnode)
            inputqslst->addtolist(inqs->current->number);

        for (outqs->current=outqs->head;
            outqs->current;
            outqs->current=outqs->current->nextnode)
            outputqslst->addtolist(outqs->current->number);

            for (inputqslst->current = inputqslst->head;inputqslst->current;inputqslst->current=inputqslst-
>current->nextnode){
             tempqueue=gqueuelist->getqueue(inputqslst->current->number);
             tq2=gnodelist->getnode(tempqueue->nodein);

             if (tq2->ntype==instruction){
              tcq2=getnode(tempqueue->nodein);
              if(!(tcq2->indexed))
                if (tcq2->finishexec>tcq1->startexec)
                  assign_indices(tcq2->nodeid,index+1);
                else
                  assign_indices(tcq2->nodeid,index);
             };

        };
```

```
      for (outputqslst->current = outputqslst->head;outputqslst->current;outputqslst->current=outputqslst-
>current->nextnode){
          tempqueue=gqueuelist->getqueue(outputqslst->current->number);
          tq2=gnodelist->getnode(tempqueue->nodeout);
          if (tq2->ntype==instruction){
            tcq2=getnode(tempqueue->nodeout);
            if((!(tcq2->indexed))||tcq2->index>=tcq1->index)
              if (tcq2->startexec<tcq1->finishexec)
                assign_indices(tcq2->nodeid,index-1);
              else
                assign_indices(tcq2->nodeid,index);
          };
        };
       };
      };


    cnode* cylinder::find_latest(double t){
      double max=0.0;
      cnode* temp;

      for (current=head;current;current=current->nextslice)
        for(current->nodelist->current=current->nodelist->head;
          current->nodelist->current;
          current->nodelist->current=current->nodelist->current->nextitem){
          if (t==0){
            if(current->nodelist->current->element->finishexec > max){
              max = current->nodelist->current->element->finishexec;
              temp =  current->nodelist->current->element;
            };
          }
          else{
            if((current->nodelist->current->element->finishexec <= t)&&
              (current->nodelist->current->element->finishexec > max)){
              max = current->nodelist->current->element->finishexec;
              temp =  current->nodelist->current->element;
            };
          };
        };
        if (max==0)
          for (current=head;current;current=current->nextslice)
```

```
        for(current->nodelist->current=current->nodelist->head;
          current->nodelist->current;
         current->nodelist->current=current->nodelist->current->nextitem)
           if(current->nodelist->current->element->finishexec > max){
             max = current->nodelist->current->element->finishexec;
            temp =  current->nodelist->current->element;
            };


   return temp;
};


cnode* cylinder::find_latest_start(double t){
   double max=0.0;
   cnode* temp;

   for (current=head;current;current=current->nextslice)
     for(current->nodelist->current=current->nodelist->head;
        current->nodelist->current;
        current->nodelist->current=current->nodelist->current->nextitem){
        if (t==0){
          if(current->nodelist->current->element->startexec > max){
            max = current->nodelist->current->element->startexec;
            temp =  current->nodelist->current->element;
          };
        }
        else{
          if((current->nodelist->current->element->startexec < t)&&
            (current->nodelist->current->element->startexec > max)){
            max = current->nodelist->current->element->startexec;
            temp =  current->nodelist->current->element;
          };
        };
     };

   return temp;
};


void cylinder::start_after_finish(int gnum){
```

```
int i,
   j;
intlist* nolist;
cnode *tempcnode1,
    *tempcnode2;
arcnode* temparc;
arclist* arclst;
fstream arcdata,datafile;
arcdata.open("tokens",ios::out);
arclst = new arclist;
nolist = new intlist;

for (current=head;current;current=current->nextslice)
  for(current->nodelist->current=current->nodelist->head;
     current->nodelist->current;
     current->nodelist->current=current->nodelist->current->nextitem)
     nolist->addtolist(current->nodelist->current->element->nodeid);

for (nolist->current=nolist->head;nolist->current;
    nolist->current=nolist->current->nextnode){
    tempcnode1=getnode(nolist->current->number);
    i = tempcnode1->index;
    tempcnode2=find_latest(tempcnode1->startexec);
    j = tempcnode2->index;
    if(!(arclst->is_already_exist(tempcnode1->nodeid,tempcnode2->nodeid))){
       if ((tempcnode1->startexec==0)||
          (tempcnode2->finishexec==cyl_circum))
           j = j - 1;
       temparc = new arcnode;
       temparc->sourcenodeid=tempcnode2->nodeid;
       temparc->sinknodeid  =tempcnode1->nodeid;
       if (i >= j){
          temparc->initial_length=i-j;
          temparc->threshold = 1;
          temparc->consumption= 1;
          temparc->production=1;
          temparc->capacity =100;
       }
       else
        if( i < j){
          temparc->initial_length=0;
```

```
            temparc->threshold    =j-i+1;
            temparc->consumption  =1;
            temparc->production   =1;
            temparc->capacity     =100;

          };
          arclst->addtolist(temparc);
        };
    };


    gnum++;
    datafile.open("simdata",ios::outlios::app);
    for (arclst->current=arclst->head;
        arclst->current;
        arclst->current=arclst->current->nextitem){
        datafile << gnum<< "  ";
        datafile << 0 << "   ";
        datafile << arclst->current->element->sourcenodeid << "   ";
        datafile << arclst->current->element->sinknodeid << "       ";
        datafile << arclst->current->element->threshold << "   ";
        datafile << arclst->current->element->initial_length << "   ";
        datafile << arclst->current->element->consumption << "   ";
        datafile << 100<< endl;

        gnum++;

        arcdata << arclst->current->element->sourcenodeid << "  ";
        arcdata << arclst->current->element->sinknodeid << "  ";
        arcdata << arclst->current->element->initial_length << "  ";
        arcdata << arclst->current->element->threshold << "  ";
        arcdata << arclst->current->element->consumption << endl;
    };
    arcdata.close();
    datafile.close();


};


void cylinder::start_after_start(int gnum){
```

```
int i,
   j;
intlist* nolist;
cnode *tempcnode1,
    *tempcnode2;
arcnode* temparc;
arclist* arclst;
fstream arcdata,datafile;
arcdata.open("tokens",ios::out);
arclst = new arclist;
nolist = new intlist;

for (current=head;current;current=current->nextslice)
  for(current->nodelist->current=current->nodelist->head;
     current->nodelist->current;
     current->nodelist->current=current->nodelist->current->nextitem)
     nolist->addtolist(current->nodelist->current->element->nodeid);

for (nolist->current=nolist->head;nolist->current;
    nolist->current=nolist->current->nextnode){
    tempcnode1=getnode(nolist->current->number);
    i = tempcnode1->index;
    tempcnode2=find_latest_start(tempcnode1->startexec);
    j = tempcnode2->index;
    if(!(arclst->is_already_exist(tempcnode1->nodeid,tempcnode2->nodeid))){
       if (tempcnode1->startexec==0)
          j = j - 1;
       temparc = new arcnode;
       temparc->sourcenodeid=tempcnode1->nodeid;
       temparc->sinknodeid  =tempcnode2->nodeid;
       if (i > j){
          temparc->initial_length=0;
          temparc->threshold = 1;
          temparc->consumption = 1;
          temparc->production = 1;
          temparc->capacity = i-j;
       }
       else
        if(i<j){
         temparc->initial_length=j-i+1;
```

```
        temparc->threshold    =1;
        temparc->consumption  =1;
        temparc->production   =1;
        temparc->capacity     =1;
        }
       else{
        temparc->initial_length=1;
        temparc->threshold    =1;
        temparc->consumption  =1;
        temparc->production   =1;
        temparc->capacity     =1;
        };
      arclst->addtolist(temparc);
   };
};


gnum++;
datafile.open("simdata",ios::outlios::app);
for (arclst->current=arclst->head;
    arclst->current;
    arclst->current=arclst->current->nextitem){
    datafile << gnum<< "  ";
    datafile << 0 << "   ";
    datafile << arclst->current->element->sourcenodeid << "   ";
    datafile << arclst->current->element->sinknodeid << "       ";
    datafile << arclst->current->element->threshold << "   ";
    datafile << arclst->current->element->initial_length << "   ";
    datafile << arclst->current->element->consumption << "   ";
    datafile << arclst->current->element->capacity<< endl;

    gnum++;

    arcdata << arclst->current->element->sourcenodeid << "  ";
    arcdata << arclst->current->element->sinknodeid << "  ";
    arcdata << arclst->current->element->initial_length << "  ";
    arcdata << arclst->current->element->threshold << "  ";
    arcdata << arclst->current->element->consumption << endl;
};
arcdata.close();
datafile.close();
```

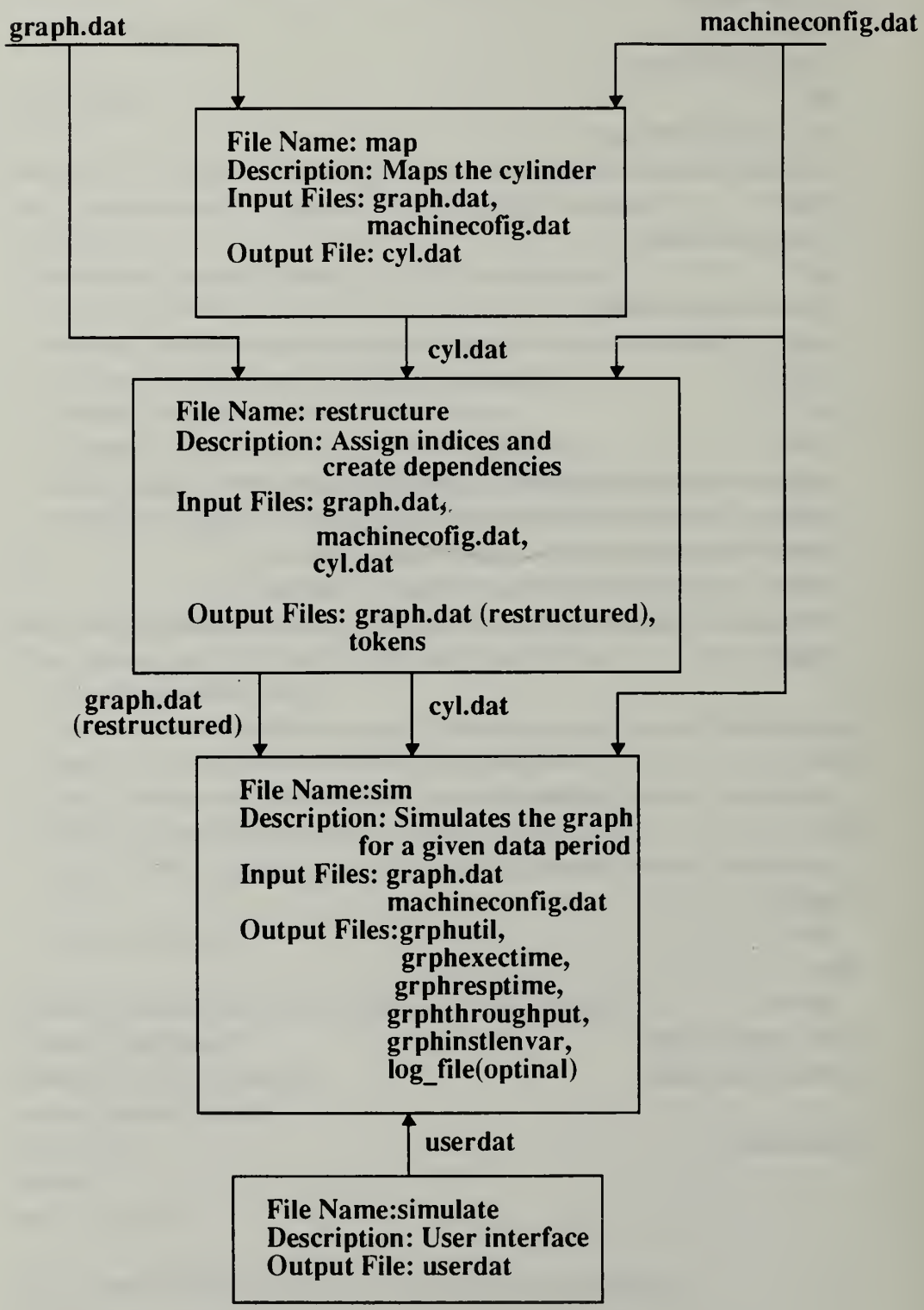
```
};


void cylinder::find_RC_arcs(int gnum){
   int choice;

   cout << "CHOICE ONE OF THE FOLLOWING\n\n";
   cout << "1..START AFTER FINISH (SAF)\n";
   cout << "2..START AFTER START  (SAS)\n\n";
   cout << "Choice : ";
   cin >> choice;
   if (choice == 1)
      start_after_finish(gnum);
   else
      start_after_start(gnum);
};




main(){
   int numassignednodes,
      temp,
      dummy,
      loop,
      loop2;
   mlist * ml;
   ml = new mlist;
   plist * pl;
   pl = new plist;
   gnodelist = new nlist;
   gqueuelist= new Qlist;
   gnode * tempnode;
   cnode * tempcnode;
   cylinder::slice * tempslice;
   clist * templist;
```

```
cylinder *c = new cylinder;
int gnum;

fstream datafile,config,cyldata,pic,mem;
datafile.open("simdata",ios::in);
config.open("machine",ios::in);
cyldata.open("cyl.map",ios::in);
mem.open("memodules",ios::in);
nummemory = ml->loadmemory(config);
numprocs = pl->loadprocessors(config);
dummy = gnodelist->loadnodes(datafile,mem,5);
gnum=gqueuelist->loadqueues(datafile,mem,gnodelist);
cout << *gnodelist;
datafile.close();
mem.close();
gnodelist->sort_topologically(gqueuelist);

tempnode = new gnode;
for (loop = 0; loop < numprocs;loop++){
   cyldata >> numassignednodes;
   tempslice = new cylinder::slice;
   templist = new clist;

   for (loop2 = 0;loop2 < numassignednodes;loop2++){
      cyldata >> temp;
      tempnode = new gnode;
      tempcnode= new cnode;
      tempnode = gnodelist->getnode(temp);
      tempcnode->nodeid = tempnode->nodeid;
      tempcnode->order  = tempnode->order;
      tempcnode->exectime= tempnode->exectime;
      templist->insert(tempcnode);
   };

   tempslice->slicenum = loop+1;
   tempslice->nodelist = templist;
   tempslice->nextslice= NULL;
   if (c->head == NULL)
      c->head=c->current=tempslice;
   else{
      c->current->nextslice = tempslice;
```

```
      c->current = tempslice;
    };
};
cyldata>>cyl_circum;

cyldata.close();

double time;
for (c->current=c->head;c->current;c->current=c->current->nextslice){
  time =0.0;
  for(c->current->nodelist->current=c->current->nodelist->head;
    c->current->nodelist->current;
    c->current->nodelist->current=c->current->nodelist->current->nextitem){
     c->current->nodelist->current->element->indexed=false;
     c->current->nodelist->current->element->index=0;
     c->current->nodelist->current->element->startexec=time;
     time=c->current->nodelist->current->element->exectime+time;
     c->current->nodelist->current->element->finishexec=time;
  };
};
for (c->current=c->head;c->current;c->current=c->current->nextslice){
  for(c->current->nodelist->current=c->current->nodelist->head;
    c->current->nodelist->current;
    c->current->nodelist->current=c->current->nodelist->current->nextitem){
     cout<<c->current->nodelist->current->element->nodeid<< " ";
     cout<<c->current->nodelist->current->element->index<<" ";
     cout<< c->current->nodelist->current->element->startexec<<" ";
     cout<< c->current->nodelist->current->element->finishexec<<" ";
  };
     cout << endl<<endl;
};
c->assign_indices(c->head->nodelist->head->element->nodeid,0);
c->find_RC_arcs(gnum);
cyldata.open("cyl.map",ios::out);
pic.open("picture",ios::out);
int level1=0;
int alt=1;
int previous = 0;

for (c->current=c->head;c->current;c->current=c->current->nextslice){
  for(c->current->nodelist->current=c->current->nodelist->head;
```

```
    c->current->nodelist->current;
    c->current->nodelist->current=c->current->nodelist->current->nextitem){
    pic<<previous<<" ";
    if (alt)
      pic<< level1<<endl;
    else
      pic<< level1+100<<endl;
    pic<<int(c->current->nodelist->current->element->finishexec/1000)<<" ";
    if (alt)
      pic<< level1<<endl;
    else
      pic<< level1+100<<endl;
    if (alt)
       alt = 0;
    else
       alt = 1;
    cyldata<<c->current->nodelist->current->element->nodeid<< " ";
    cyldata<<c->current->nodelist->current->element->index<<" ";
    cyldata<< c->current->nodelist->current->element->startexec<<" ";
    cyldata<< c->current->nodelist->current->element->finishexec<<" ";
    previous =int(c->current->nodelist->current->element->finishexec/1000);
    cyldata<<"   ";
   };
   previous = 0;
   if(alt)
     pic << 0 << " "<<level1+100<< endl;
   else
     pic << 0 << " "<<level1 << endl;
   if (alt)
      alt = 0;
   else
      alt = 1;
   level1 = level1 + 100;
   cyldata<<endl<<endl;
  };
 cyldata.close();

};
```

APPENDIX D: Interrelation of the Files in PIPDAFS and GR



LIST OF REFERENCES

[BELL 92] Bell, Harold A., A Compile-Time Approach for Chaining and Execution Control in the AN/UYS-2 Parallel Signal Processor, Master's Thesis, Naval Postgraduate School, Monterey California, June 1992.

[HSU 86] Hsu Y. P., Highly Concurrent Scalar Processing, Phd Thesis, University of Illinois at Urbana-Champaign, Urbana, Illinois, December 1986.

[KARP 66] Karp, R. M., and Miller, R. E., "Properties of a Model for Parallel Computations: Determinacy, Terminacy, Termination, Queueing," SIAM Journal of Applied Mathematics, v.14, No.6 November 1966.

[LEE 89] Lee, E. A., Wai-hung Ho, Edwin E. Goei, Jefferey C. Bier, and Shuvra Bhattacharyya, "Gabriel: A Design Environment for DSP, "IEEE Transactions on Acoustics Speech and Signal Processing," v.37, No 11,pp. 1751-1762, November 1989.

[LEE 90] Lee, E. A., Bier, J. C., "Architectures for Statically Scheduled Dataflow," Journal of Parallel and Distributed Computing, v. 10, pp. 333-348, December 1990.

[LIT 91] Little, B. S., "A Technique for Predictable Real-Time Execution in the AN/UYS-2 Parallel Signal Processing Architecture," Master's Thesis, Naval Postgraduate School, Monterey, California, December 1990.

[POPS 90] AT&T Technologies, Report 58854401, "Enhanced Modular Signal Processor (EMSP) Principles of Operation (POPS)," AT&T Bell Laboratories, March 1990.

[RAU 81] Rau, B.R., Kuekes, P.J. and Glaeser, C.D. "A Statistically Scheduled VLSI Interconnect for Parallel Processors," in VLSI Systems and Computations, Computer Science Press. pp. 389-395,1981.

[SLZ 92] Shukla, S. B., Little, B. S., and Zaky, A., "A Compile-time Technique for Controlling Real-time Execution of Task-level Data-flow Graphs," presented at the 1992 International Conference on Parallel Processing.

INITIAL DISTRIBUTION LIST

Defense Technical Information Center — 2
Cameron Station
Alexandria, VA 22304-6145

Dudley Knox Library, Code 052 — 2
Naval Postgraduate School
Monterey, CA 93943

Chairman, Code 37 CS — 1
Computer Science Department
Naval Postgraduate School
Monterey, CA 93943

Dr. Amr Zaky, Code CS/Za — 2
Professor, Computer Science Department
Naval Postgraduate School
Monterey, CA 93943-5000

Dr. Shridhar Shukla, Code EC/Sh — 2
Professor, Department of Electrical and Computer Engineering
Naval Postgraduate School
Monterey, CA 93943-5000

Deniz Kuvvetleri Komutanligi — 2
Personel Daire Baskanligi
Bakanliklar, Ankara / TURKEY

Golcuk Tersanesi Komutanligi — 1
Golcuk, Kocaeli / TURKEY

Deniz Harp Okulu Komutanligi — 1
81704 Tuzla, Istanbul / TURKEY

Taskizak Tersanesi Komutanligi — 1
Kasimpasa, Istanbul / TURKEY

LTjg Cem AKIN — 1
Haciyusuf mah. I. okul sok.
Esen apt. No:27 kat:2 daire:3
10200 Bandirma / TURKEY